Efficient Implementation of Iterative Polynomial Matrix EVD Algorithms Exploiting Structural Redundancy and Parallelisation

Fraser K. Coutts, *Student Member, IEEE*, Ian K. Proudler, Stephan Weiss, *Senior Member, IEEE*

***Abstract*—A number of algorithms are capable of iteratively calculating a polynomial matrix eigenvalue decomposition (PEVD), which is a generalisation of the EVD and will diagonalise a parahermitian polynomial matrix via paraunitary operations. While offering promising results in various broadband array processing applications, the PEVD has seen limited deployment in hardware due to the high computational complexity of these algorithms. Akin to low complexity divide-and-conquer (DaC) solutions to eigenproblems, this paper addresses a partially parallelisable DaC approach to the PEVD. A novel algorithm titled parallel-sequential matrix diagonalisation exhibits significantly reduced algorithmic complexity and run-time when compared with existing iterative PEVD methods. The DaC approach, which is shown to be suitable for multi-core implementation, can improve eigenvalue resolution at the expense of decomposition mean squared error, and offers a trade-off between the approximation order and accuracy of the resulting paraunitary matrices.**

***Index Terms*—parahermitian matrix, paraunitary matrix, polynomial matrix eigenvalue decomposition, parallel, algorithm.**

I. Introduction

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 captures 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 $z \in \mathbb{C}$ [3], [4].

A polynomial matrix eigenvalue decomposition (PEVD) has been defined as an extension of the EVD to parahermitian polynomial matrices in [5]. The PEVD uses finite impulse response (FIR) paraunitary matrices [6] to approximately diagonalise and typically spectrally majorise [7] a CSD matrix and its associated space-time covariance matrix. Recent work in [8], [9] 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 [10] 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 can be obtained via the PEVD. For example, the PEVD has been successfully used in broadband MIMO precoding and equalisation using linear [11]–[16] and non-linear [17], [18] approaches, broadband angle of arrival estimation [19]–[22], broadband beamforming [23]–[25], optimal subband coding [7], [26], joint source-channel coding [27], source separation [28], and scene discovery [29].

Existing PEVD algorithms include second-order sequential best rotation (SBR2) [5], sequential matrix diagonalisation (SMD) [30], and various evolutions of both algorithm families [31]–[33]. Different from fixed order time domain PEVD schemes in [34], [35] and DFT-based approaches in [36]–[38], the SBR2 and SMD algorithm families have proven convergence. Both SBR2 and SMD algorithms employ iterative time domain schemes to approximately diagonalise a parahermitian matrix, and encourage — or even guarantee [39] — spectral majorisation such that the power spectral densities (PSDs) of the resulting eigenvalues are ordered at all frequencies [7].

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 [40]–[42], but the SMD algorithm, which can achieve superior levels of diagonalisation [30], has been restricted to software applications due to its high computational complexity 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 [5], [43]–[45], which translates directly into a growth of computational complexity. By applying a row-shift truncation scheme for paraunitary matrices in [45]–[47], the polynomial order can be reduced with little loss to paraunitarity of the eigenvectors. These efforts, including a low cost cyclic-by-row numerical approximation of the EVD [48], [49] and optimisation over reduced parameter sets [33], [49], [50] have nonetheless not been able to reduce computational cost sufficiently to invite a hardware realisation.

Therefore, this paper attempts to reduce the computational

This work was supported in parts by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/S000631/1 and the MOD University Defence Research Collaboration in Signal Processing. Fraser Coutts was the recipient of a Caledonian Scholarship by the Carnegie Trust.

F.K. Coutts is with the Institute for Digital Communications, School of Engineering, University of Edinburgh, Edinburgh, Scotland (e-mail fcoutts@exseed.ed.ac.uk).

I.K. Proudler and S. Weiss are with the Centre for Signal & Image Processing (CeSIP), Department of Electronic & Electrical Engineering, University of Strathclyde, Glasgow G1 1XW, Scotland (e-mail {ian.proudler,stephan.weiss}@strath.ac.uk).

cost of SMD-type algorithms through a novel combination and extension of recent numerical optimisation approaches. The structural redundancy inside the parahermitian matrix, i.e. its inherent symmetry, can be exploited in order to reduce both computations and memory requirements [51]. A divide-and-conquer (DaC) PEVD algorithm in [52] segments a large parahermitian matrix into multiple independent parahermitian matrices, which are subsequently diagonalised independently and simultaneously, demonstrating promising performance characteristics in applications [22], [47]. In the approach presented in this paper, both the 'divide' and 'conquer' stages make use of algorithmic improvements from [51] and [53], and the truncation schemes from [44], to minimise algorithm complexity. The final stage of the algorithm employs a novel variant of the row-shift truncation scheme of [45] to reduce the polynomial order of the paraunitary matrix.

Below, Sec. II will provide a summary of the notations and definitions used throughout this paper. Sec. III will introduce polynomial matrix truncation schemes used within the proposed parallel-sequential matrix diagonalisation (PSMD) approach. The PSMD algorithm approach is outlined in Sec. IV, and performance metrics are defined in Sec. V. Simulation results for PSMD are compared to existing iterative PEVD methods in Sec. VI, with comments on hardware implementations in Sec. VII and conclusions drawn in Sec. VIII.

II. Notations and Definitions

In this paper, upper- and lowercase boldface variables, such as $\mathbf{A}$ and $\mathbf{a}$, refer to matrix- and vector-valued quantities, respectively. A dependency on a continuous and discrete variable is indicated via brackets and parentheses, respectively, such as $\mathbf{A}(t)$, $t \in \mathbb{R}$, or $\mathbf{a}[n]$, $n \in \mathbb{Z}$. Polynomial quantities are denoted by their dependency on z and italic font, such as $\boldsymbol{A}(z)$. The expectation operator is denoted as $\mathcal{E}\{\cdot\}$ and $\{\cdot\}^{\rm H}$ indicates a Hermitian transpose. The parahermitian conjugate $\{\cdot\}^{\rm P}$ implies a Hermitian transpose and a time reversal, such that $\boldsymbol{R}^{\rm P}(z) = \boldsymbol{R}^{\rm H}(1/z^*)$ [4].

In a broadband array scenario, a space-time covariance matrix

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

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. 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 $\mathbf{R}[\tau] = \mathbf{R}^{\rm H}[-\tau]$. The CSD matrix $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M\times M}$ arises as the z-transform of a space-time covariance matrix $\mathbf{R}[\tau]$, NO FORMULA $\boldsymbol{R}(z) = \sum_{\tau=-T}^{T} \mathbf{R}[\tau]z^{-\tau}$, where T is the maximum lag of $\mathbf{R}[\tau]$; i.e., $\mathbf{R}[\tau] = \mathbf{0}\ \forall\ |\tau| > T$. The relationship between time domain and transform domain quantities is abbreviated below as $\boldsymbol{R}(z) \bullet\!\!-\!\!\circ\ \mathbf{R}[\tau]$. Since $\mathbf{R}[\tau] = \mathbf{R}^{\rm H}[-\tau]$, $\boldsymbol{R}(z)$ is a parahermitian matrix, such that $\boldsymbol{R}(z) = \boldsymbol{R}^{\rm P}(z)$ [4]. The PEVD [5] uses a paraunitary matrix $\boldsymbol{F}(z)$ to approximately diagonalise a parahermitian CSD matrix $\boldsymbol{R}(z)$ such that

$$\boldsymbol{R}(z) \approx \boldsymbol{F}^{\rm P}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)\ , \qquad (1)$$

where $\boldsymbol{D}(z) \approx {\rm diag}\{\boldsymbol{D}_1(z),\ \boldsymbol{D}_2(z),\ \ldots,\ \boldsymbol{D}_M(z)\}$ approximates a diagonal matrix and is typically spectrally majorised with PSDs $\boldsymbol{D}_i(e^{{\rm j}\Omega}) \geq \boldsymbol{D}_{i+1}(e^{{\rm j}\Omega})\ \forall\ \Omega,\ i = 1\ldots(M-1)$, where $\boldsymbol{D}_i(e^{{\rm j}\Omega}) = \boldsymbol{D}_i(z)|_{z=e^{{\rm j}\Omega}}$. The diagonal of $\boldsymbol{D}(z)$ contains approximate polynomial eigenvalues, and the rows of $\boldsymbol{F}(z)$ are approximate polynomial eigenvectors. The paraunitary property of the eigenvectors ensures that

$$\boldsymbol{F}(z)\boldsymbol{F}^{\rm P}(z) = \boldsymbol{F}^{\rm P}(z)\boldsymbol{F}(z) = \mathbf{I}_M\ , \qquad (2)$$

where $\mathbf{I}_M$ is an $M \times M$ identity matrix. Note that the decomposition in (1) is unique up to permutations and arbitrary all-pass filters applied to the eigenvectors.

Equation (1) has only approximate equality, as the PEVD of a finite order polynomial matrix is generally transcendental, i.e. 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 [8]. A finite order approximation will therefore lead to only approximate diagonality of $\boldsymbol{D}(z)$ in (1). Similarly, a finite order approximation of $\boldsymbol{F}(z)$ through trimming will result in only approximate equality in (2).

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

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

where $\mathbf{R}[0]$ is the 'lag zero' matrix of $\boldsymbol{R}(z)$, $\boldsymbol{R}^{(+)}(z)$ contains terms for positive lag elements only, and $\boldsymbol{R}^{(-)}(z) = \boldsymbol{R}^{(+),{\rm P}}(z)$ [51]. 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 paper, we use the notation $\overline{\{\cdot\}}$ 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

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

If we possess knowledge of $\overline{\boldsymbol{R}}(z)$, we have all the information required to obtain $\boldsymbol{R}(z)$, $\overline{\mathbf{R}}[\tau]$, and $\mathbf{R}[\tau]$. Given this relationship, we therefore refer to $\overline{\boldsymbol{R}}(z)$ as a parahermitian matrix throughout this paper for brevity. In addition, we refer to $\overline{\boldsymbol{R}}(z)$ and $\overline{\mathbf{R}}[\tau]$ as the 'half-matrix' versions of the 'full-matrix' representations $\boldsymbol{R}(z)$ and $\mathbf{R}[\tau]$, respectively.

Throughout this paper, $\mathbf{R}_{m,k}[\tau] \circ\!\!-\!\!\bullet\ \boldsymbol{R}_{m,k}(z)$ represents the element in the mth row and kth column of $\mathbf{R}[\tau] \circ\!\!-\!\!\bullet\ \boldsymbol{R}(z)$.

III. Polynomial Matrix Truncation Schemes

A. State-of-the-Art in Polynomial Matrix Truncation

The polynomial matrix truncation method from [44] is employed within PSMD. 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

$$f_{\rm trim}(\mathbf{Y}[\tau],\mu) = \begin{cases} \mathbf{Y}[\tau], & T_1 + T_3(\mu) \leq \tau \leq T_2 - T_4(\mu) \\ \mathbf{0}, & \text{otherwise}\ . \end{cases} \qquad (3)$$

The amount of energy lost by removing the $T_3(\mu)$ leading and $T_4(\mu)$ trailing lags of $\mathbf{Y}[\tau]$ via the $f_{\mathrm{trim}}(\cdot)$ operation is measured by

$$\gamma_{\mathrm{trim}} = 1 - \frac{\sum_\tau \|f_{\mathrm{trim}}(\mathbf{Y}[\tau], \mu)\|_{\mathrm{F}}^2}{\sum_\tau \|\mathbf{Y}[\tau]\|_{\mathrm{F}}^2} , \qquad (4)$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. A parameter μ is used to provide an upper bound for γ_{trim}. Given the above, the truncation procedure can be expressed as the constrained optimisation problem:

$$\text{maximise } (T_3(\mu) + T_4(\mu)) , \quad \text{s.t.} \quad \gamma_{\mathrm{trim}} \leq \mu . \qquad (5)$$

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.

B. Compensated Row-Shift Truncation Method

The row-shift truncation method [45], [46] exploits ambiguity in the paraunitary matrices [45], [54]. 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 is of the form of an integer number of unit delays. If $\boldsymbol{D}(z)$ is exactly diagonal, then this phase ambiguity may permit 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,

$$\begin{aligned}\boldsymbol{R}(z) \approx \boldsymbol{F}^{\mathrm{P}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z) &= \hat{\boldsymbol{F}}^{\mathrm{P}}(z)\boldsymbol{\Gamma}(z)\boldsymbol{D}(z)\boldsymbol{\Gamma}^{\mathrm{P}}(z)\hat{\boldsymbol{F}}(z) \\ &= \hat{\boldsymbol{F}}^{\mathrm{P}}(z)\boldsymbol{D}(z)\hat{\boldsymbol{F}}(z) , \qquad (6)\end{aligned}$$

and $\boldsymbol{D}(z)$ in unaffected. The row-shift truncation method in [45] exploits this by searching for the best delay and truncation of all eigenvector approximations in a paraunitary matrix $\boldsymbol{F}(z)$ calculated by any PEVD algorithm.

During the iterations of a PEVD algorithm, or due to large M, the diagonalisation of $\boldsymbol{D}(z)$ may be poor with significant non-zero off-diagonal elements, such that $\boldsymbol{\Gamma}(z)$ does not cancel as in (6). Since we are typically only interested in the approximate polynomial eigenvalues stored on the diagonal of $\boldsymbol{D}(z)$, we propose a compensated variation of the row-shift truncation in [45] which incorporates the matrix $\boldsymbol{\Gamma}(z)$ into the parahermitian matrix to avoid propagating the decomposition error that would otherwise arise from neglecting non-zero off-diagonal components. Define the augmented parahermitian matrix $\hat{\boldsymbol{D}}(z) = \boldsymbol{\Gamma}(z)\boldsymbol{D}(z)\boldsymbol{\Gamma}^{\mathrm{P}}(z)$, so that the decomposition accuracy can now be maintained while using the lower order $\hat{\boldsymbol{F}}(z)$, s.t. $\boldsymbol{R}(z) \approx \hat{\boldsymbol{F}}^{\mathrm{P}}(z)\hat{\boldsymbol{D}}(z)\hat{\boldsymbol{F}}(z)$. Note that because of paraunitarity of $\boldsymbol{\Gamma}(z)$, $\hat{\boldsymbol{D}}(z)$ possesses the same polynomial eigenvalues as $\boldsymbol{D}(z)$. While $\hat{\boldsymbol{D}}(z)$ may now have a higher polynomial order than $\boldsymbol{D}(z)$, a suitable choice for $\boldsymbol{\Gamma}(z)$ can lead to an order reduction of the paraunitary matrix, which is typically more important for application purposes [17], [22], [24], [26], [27].

From [45], $\boldsymbol{\Gamma}(z)$ can take the form

$$\boldsymbol{\Gamma}(z) = \mathrm{diag}\{z^{\tau_1},\ z^{\tau_2},\ \ldots,\ z^{\tau_M}\} . \qquad (7)$$

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 $\boldsymbol{f}_m(z) : \mathbb{C} \rightarrow \mathbb{C}^{1\times M}$, $m = 1 \ldots M$,

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

Each row — which has minimum lag $T_{1,m}$ and maximum lag $T_{2,m}$ — is then truncated individually according to $f_{\mathrm{trim}}(\mathbf{f}_m[\tau], \mu)$. The row shifts, τ_m, in (7) are then set equal to $(T_{1,m} + T_{3,m}(\mu))$, $m = 1 \ldots M$, 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 (3) 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 $\max\limits_{m=1\ldots M}\{T_m(\mu)\}$, where $T_m(\mu) = T_{2,m} - (T_{3,m}(\mu) + T_{4,m}(\mu))$, with $T_{3,m}(\mu)$ and $T_{4,m}(\mu)$ obtained from (5).

When applying compensated row-shift truncation (CRST) to a matrix $\boldsymbol{F}(z)$, we therefore obtain

$$[\hat{\boldsymbol{F}}(z), \hat{\boldsymbol{D}}(z)] \leftarrow f_{\mathrm{CRST}}(\boldsymbol{F}(z), \boldsymbol{D}(z), \mu) ,$$

with $\hat{\boldsymbol{F}}(z)$ having rows $\hat{\boldsymbol{f}}_m(z) \bullet\!\!-\!\!\circ\ \hat{\mathbf{f}}_m[\tau] = f_{\mathrm{trim}}(\mathbf{f}_m[\tau], \mu)$.

IV. Parallel-Sequential Matrix Diagonalisation

Motivated by the results obtained by a DaC algorithm in [22], [47], this section outlines the components of a novel parallel-sequential matrix diagonalisation (PSMD) PEVD algorithm, which is summarised in Sec. IV-A. Sec. IV-B and Sec. IV-C explain its 'divide' and 'conquer' steps, respectively. Some comments on algorithm convergence are provided in Sec. IV-D.

A. Overview

The PSMD algorithm diagonalises a parahermitian matrix $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{M\times M}$ via a number of paraunitary operations. The algorithm outputs 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 (1) is satisfied.

While the majority of iterative PEVD algorithms attempt to diagonalise an entire $M \times M$ parahermitian matrix at once, the PSMD algorithm — which improves upon the algorithm in [52] — performs two larger paraunitary steps whose effect is outlined in Fig. 1. A first paraunitary similarity transform brings the matrix into a block diagonal form in a 'divide' step. A second paraunitary similarity transform then diagonalises or 'conquers' each of the smaller, now independent, matrices on the diagonal separately. The 'divide' step is a sequential process, while the 'conquer' step can be parallelised. For example, a matrix $\boldsymbol{R}(z) : \mathbb{C} \rightarrow \mathbb{C}^{20\times 20}$ might be 'divided' into four 5×5 parahermitian matrices, each of which can be diagonalised independently and simultaneously. Fig. 1 shows the state of the parahermitian matrix at each stage of the process for this example. As in [51], here we exploit the natural symmetry of the parahermitian matrix structure and only store one half of its elements; i.e., we diagonalise $\bar{\boldsymbol{R}}(z)$.

Research in [49], [55] 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. To reduce the computational complexity, we therefore employ the restricted update method of [53] in the 'divide' and 'conquer' stages. This not only restricts the search space of the algorithms used in each stage, but also restricts the portion of the parahermitian matrix that is updated at each iteration.

If $\boldsymbol{R}(z)$ is of large spatial dimension, an algorithm named half-matrix restricted update sequential matrix segmentation (HRSMS) is repeatedly used to 'divide' the matrix into block diagonal form that contains multiple independent parahermitian matrices. This function generates a paraunitary matrix $\boldsymbol{T}(z)$ that 'divides' an input matrix $\bar{\boldsymbol{A}}(z)$ into two independent parahermitian matrices, $\bar{\boldsymbol{A}}_{11}(z)$ and $\bar{\boldsymbol{A}}_{22}(z)$, of smaller spatial dimension. $\bar{\boldsymbol{A}}_{11}(z)$ is then subject to further 'division' if it still has sufficiently large spatial dimension. Following a number of 'division' steps, each of the output independent parahermitian matrices are stored on the diagonal of matrix $\bar{\boldsymbol{R}}'(z)$; thus, $\bar{\boldsymbol{R}}'(z)$ is block diagonal by construction. The matrices $\boldsymbol{T}(z)$ are concatenated to form an overall dividing matrix $\boldsymbol{G}(z)$. It is therefore possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\boldsymbol{G}^{\mathrm{P}}(z)\boldsymbol{R}'(z)\boldsymbol{G}(z)$.

Each block on the diagonal of matrix $\bar{\boldsymbol{R}}'(z)$ is then diagonalised in parallel through the use of a half-matrix version of the algorithm from [53], named half-matrix restricted update sequential matrix diagonalisation (HRSMD). The diagonalised outputs, $\bar{\boldsymbol{C}}(z)$, are placed on the diagonal of matrix $\bar{\boldsymbol{D}}(z)$, and the corresponding paraunitary matrices, $\boldsymbol{V}(z)$, are stored on the diagonal of matrix $\boldsymbol{J}(z)$. The matrix $\boldsymbol{R}'(z)$ can be approximately reconstructed from $\boldsymbol{J}^{\mathrm{P}}(z)\boldsymbol{D}(z)\boldsymbol{J}(z)$; by extension, it is possible to approximately reconstruct $\boldsymbol{R}(z)$ from the product $\boldsymbol{G}^{\mathrm{P}}(z)\boldsymbol{J}^{\mathrm{P}}(z)\boldsymbol{D}(z)\boldsymbol{J}(z)\boldsymbol{G}(z) = \boldsymbol{F}^{\mathrm{P}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$.

The polynomial matrix truncation scheme of Sec. III-A is implemented within HRSMS and HRSMD. The paraunitary matrix compensated row-shift truncation scheme of Sec. III-B is more costly than the method of Sec. III-A, and does not provide an increase in truncation performance when implemented within the SMD algorithm [46] — which the aforementioned algorithms are based on. However, a similar strategy has been found to be effective when truncating the output paraunitary matrix of a DaC PEVD scheme in [47]. Similarly, we employ this scheme to truncate the final paraunitary matrix in PSMD.

Algorithm 1 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 δ 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 'division'. The parameter P will be discussed in subsequent sections; $\mathbf{I}_M$ and $\mathbf{0}_M$ are identity and zero matrices of spatial dimensions $M \times M$, respectively.

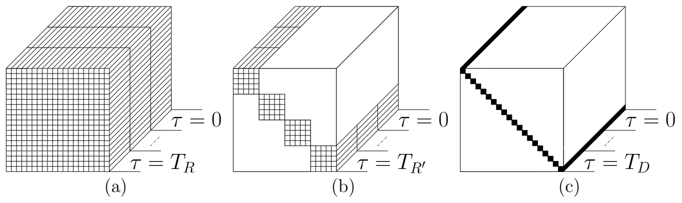


Fig. 1. Concept of PSMD: (a) original matrix $\bar{\mathbf{R}}[\tau] \in \mathbb{C}^{20\times 20}$, which in a first 'divide' paraunitary similarity transform step yields (b) the block diagonal result $\bar{\mathbf{R}}'[\tau]$; a second paraunitary similarity transform, which can now be applied to each subblock separately leads to (c) the diagonalised output $\bar{\mathbf{D}}[\tau]$.

```
Input: $\boldsymbol{R}(z)$, $\mu$, $\mu_t$, $\mu_s$, $\delta$, $\epsilon$, $I_D$, $I_C$, $\hat{M}$, $P$
Output: $\boldsymbol{D}(z)$, $\boldsymbol{F}(z)$
Determine if input matrix is large:
if $M > \hat{M}$ then
    Large matrix — 'divide-and-conquer':
    $M' \leftarrow M$, $\bar{\boldsymbol{A}}(z) \leftarrow \bar{\boldsymbol{R}}(z)$, $\boldsymbol{G}(z) \leftarrow \mathbf{I}_M$,
      $\bar{\boldsymbol{R}}'(z)$,$\boldsymbol{J}(z)$,$\bar{\boldsymbol{D}}(z) \leftarrow \mathbf{0}_M$,  $\alpha \leftarrow 0$
    'Divide' matrix:
    while $M' > \hat{M}$ do
        $\alpha \leftarrow \alpha + 1$
        $[\bar{\boldsymbol{A}}_{11}(z), \bar{\boldsymbol{A}}_{22}(z), \boldsymbol{T}(z)] \leftarrow$
          HRSMS($\bar{\boldsymbol{A}}(z), I_D, P, \delta, \mu, \mu_t$)
        $(M - M')$ ones appended to lag zero diagonal
          of $\boldsymbol{T}(z)$ to form $\hat{\boldsymbol{T}}(z)$
        Store $\bar{\boldsymbol{A}}_{22}(z)$ on diagonal of $\bar{\boldsymbol{R}}'(z)$ in $\alpha$th
          $P \times P$ block from bottom-right
        $\boldsymbol{G}(z) \leftarrow \hat{\boldsymbol{T}}(z)\boldsymbol{G}(z)$, $\bar{\boldsymbol{A}}(z) \leftarrow \bar{\boldsymbol{A}}_{11}(z)$,
          $M' \leftarrow M' - P$
    end
    Store $\bar{\boldsymbol{A}}(z)$ on diagonal of $\bar{\boldsymbol{R}}'(z)$ in top-left
      $M' \times M'$ block
    'Conquer' independent matrices (in parallel):
    for $\gamma \leftarrow 1$ to $(\alpha + 1)$ do
        $\bar{\boldsymbol{B}}(z)$ is $\gamma$th block of $\bar{\boldsymbol{R}}'(z)$ from bottom-right
        $[\bar{\boldsymbol{C}}(z), \boldsymbol{V}(z)] \leftarrow$ HRSMD($\bar{\boldsymbol{B}}(z), I_C, \epsilon, \mu, \mu_t$)
        Store $(\bar{\boldsymbol{C}}(z), \boldsymbol{V}(z))$ in $\gamma$th block of
          $(\bar{\boldsymbol{D}}(z), \boldsymbol{J}(z))$ from bottom-right
    end
    $\boldsymbol{F}(z) \leftarrow \boldsymbol{J}(z)\boldsymbol{G}(z)$
else
    Small matrix — perform HRSMD only:
    $[\boldsymbol{F}(z), \bar{\boldsymbol{D}}(z)] \leftarrow$ HRSMD($\bar{\boldsymbol{R}}(z), I_C, \epsilon, \mu, \mu_t$)
end
$\mathbf{D}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{D}[\tau], \mu)$
Apply compensated row-shift truncation:
$[\boldsymbol{F}(z), \boldsymbol{D}(z)] \leftarrow f_{\mathrm{shift}}(\boldsymbol{F}(z), \boldsymbol{D}(z), \mu_s)$
```

Algorithm 1: PSMD Algorithm

B. 'Dividing' the Parahermitian Matrix

When $\boldsymbol{R}(z)$ is measured to have spatial dimension $M > \hat{M}$, the 'divide' stage of PSMD comes into effect. This stage recursively applies half-matrix restricted update sequential matrix segmentation (HRSMS) to 'divide' $\boldsymbol{R}(z)$ into multiple independent parahermitian matrices. HRSMS is a novel

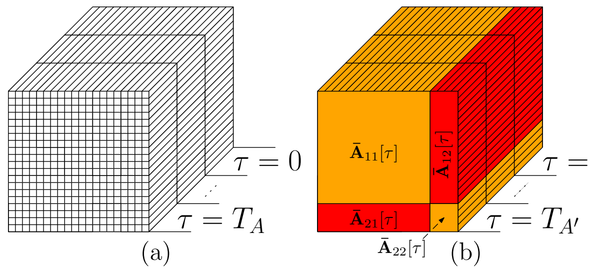

$\tau = 0$
$\tau = T_A$
(a)
$\bar{\mathbf{A}}_{11}[\tau]$
$\bar{\mathbf{A}}_{12}[\tau]$
$\bar{\mathbf{A}}_{21}[\tau]$
$\bar{\mathbf{A}}_{22}[\tau]$
$\tau = 0$
$\tau = T_{A'}$
(b)

Fig. 2. (a) Original matrix $\bar{\mathbf{A}}[\tau] \in \mathbb{C}^{20\times 20}$ with regions to be driven to zero in HRSMS, and (b) segmented result for $P = 4$; T_A and $T_{A'}$ are the maximum lags for the original and segmented matrices, respectively.

variant of SMD [30] designed to segment a matrix $\bar{\boldsymbol{A}}(z) : \mathbb{C} \to \mathbb{C}^{M'\times M'}$ into two independent parahermitian matrices $\bar{\boldsymbol{A}}_{11}(z) : \mathbb{C} \to \mathbb{C}^{(M'-P)\times(M'-P)}$ and $\bar{\boldsymbol{A}}_{22}(z) : \mathbb{C} \to \mathbb{C}^{P\times P}$, and two matrices $\bar{\boldsymbol{A}}_{12}(z) : \mathbb{C} \to \mathbb{C}^{(M'-P)\times P}$ and $\bar{\boldsymbol{A}}_{21}(z) : \mathbb{C} \to \mathbb{C}^{P\times(M'-P)}$, where $\boldsymbol{A}_{12}(z) = \boldsymbol{A}_{21}^{\mathrm{P}}(z)$ approximates a matrix of zeroes. The dimensions of the smaller matrix produced during division, P, is forced to satisfy $P \le \hat{M}$. Each instance of HRSMS is provided with a parameter I_D — which defines the maximum possible number of algorithm iterations — a stopping threshold δ, and truncation parameters μ and μ_t.

To achieve matrix segmentation, the HRSMS algorithm uses a series of elementary paraunitary operations to iteratively minimise the energy in the bottom-left $P \times (M' - P)$ and top-right $(M' - P) \times P$ regions of $\bar{\boldsymbol{A}}(z)$. Each elementary paraunitary operation consists of two steps: first a delay step is used to move the region with the largest energy to lag zero; then an EVD diagonalises the lag zero matrix, transferring the shifted energy onto the diagonal.

We employ the restricted update method of [53] in HRSMS, 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. This restriction limits both the number of search operations and the computations required to update the increasingly segmented 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.

Fig. 2 illustrates the segmentation process of HRSMS for $M' = 20$ and $P = 4$. In this example, if $M' - P = 16$ is greater than $\hat{M}$, $\bar{\mathbf{A}}_{11}[\tau]$ will be subject to further division.

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

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

$$\boldsymbol{S}^{(i)}(z) = \boldsymbol{U}^{(i)}(z)\boldsymbol{S}^{(i-1)}(z)\boldsymbol{U}^{(i),\mathrm{P}}(z)$$
$$\boldsymbol{H}^{(i)}(z) = \boldsymbol{U}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$$

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

$$\boldsymbol{U}^{(i)}(z) = \mathbf{Q}^{(i)}\boldsymbol{\Lambda}^{(i)}(z)\ , \tag{8}$$

and I is defined later. The product in (8) consists of a paraunitary delay matrix

$$\boldsymbol{\Lambda}^{(i)}(z) = \mathrm{diag}\{\underbrace{1\ \ldots\ 1}_{M'-P}\ \underbrace{z^{\tau^{(i)}}\ \ldots\ z^{\tau^{(i)}}}_{P}\} \tag{9}$$

and a unitary $\mathbf{Q}^{(i)}$, with the result that $\boldsymbol{U}^{(i)}(z)$ in (8) is paraunitary. For subsequent discussion, it is convenient to define intermediate variables $\bar{\boldsymbol{S}}^{(i)\prime}(z)$ and $\boldsymbol{H}^{(i)\prime}(z)$ where

$$\begin{aligned}\bar{\boldsymbol{S}}^{(i)\prime}(z) &= f_{\mathrm{shift,SMS}}(\,\bar{\boldsymbol{S}}^{(i-1)}(z),\ \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}\,)\\ \boldsymbol{H}^{(i)\prime}(z) &= \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)\quad ,\end{aligned} \tag{10}$$

where $f_{\mathrm{shift,SMS}}(\cdot)$ — which is described in Algorithm 2 — 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 $\bar{\mathbf{S}}^{(i-1)}[\tau]$. Matrix $\boldsymbol{\Lambda}^{(i)}(z)$ is selected based on the position of the dominant region in $\bar{\boldsymbol{S}}^{(i-1)}(z)$ •—○ $\bar{\mathbf{S}}^{(i-1)}[\tau]$, as identified by the parameter set

$$\tau^{(i)} = \arg\max_{\tau}\left\{\|\bar{\mathbf{S}}_{21}^{(i-1)}[\tau]\|_{\mathrm{F}}\ ,\ \|\bar{\mathbf{S}}_{12}^{(i-1)}[-\tau]\|_{\mathrm{F}}\right\}\ , \tag{11}$$

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

Algorithm 2: $f_{\mathrm{shift,SMS}}(\cdot)$ function

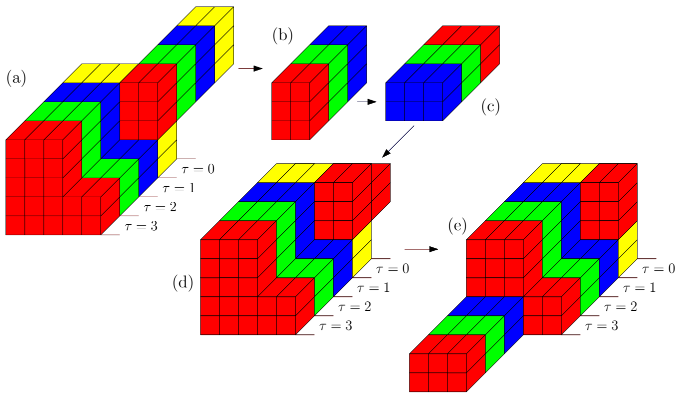


Fig. 3. Example for a matrix where in the ith iteration the Frobenius norm of a region in the top-right of the matrix is maximum: (a) the region is shifted (here in negative direction), with elements in the region past lag zero (b) extracted and (c) parahermitian conjugated; (d) these elements are appended to the far (hidden) bottom-left region at lag zero and (e) shifted in the opposite (here positive) direction.

where

$$\|\bar{\mathbf{S}}_{21}^{(i-1)}[\tau]\|_{\mathrm{F}} = \sqrt{\sum_{m=M'-P+1}^{M'} \sum_{k=1}^{M'-P} |\bar{\mathbf{S}}_{m,k}^{(i-1)}[\tau]|^2} \; ,$$

and the norm of $\|\bar{\mathbf{S}}_{12}^{(i-1)}[\tau]\|_{\mathrm{F}}$ is similarly calculated over the terms $\bar{\mathbf{S}}_{k,m}^{(i-1)}[\tau]$.

According to the $f_{\text{shift,SMS}}(\cdot)$ function: if (11) returns $\tau^{(i)} > 0$, then the bottom-left $P \times (M' - P)$ region of $\bar{\boldsymbol{S}}^{(i-1)}(z)$ 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 (i.e., Hermitian transposed and time reversed) 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 $|\tau^{(i)}|$ elements towards increasing τ. Note that the $f_{\text{shift,SMS}}(\cdot)$ function shifts the bottom-right $P \times P$ region of $\bar{\boldsymbol{S}}^{(i-1)}(z)$ in opposite directions, such that this region remains unaffected. An efficient implementation of $f_{\text{shift,SMS}}(\cdot)$ can therefore exclude this region from shifting operations.

An efficient example of the shift operation is depicted in Fig. 3 for the case of $\bar{\boldsymbol{S}}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{5\times5}$ 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 (10) moves the dominant bottom-left or top-right region in $\bar{\mathbf{S}}^{(i-1)}[\tau]$ into the lag zero coefficient matrix $\bar{\mathbf{S}}^{(i)\prime}[0]$. In accordance with the restricted update scheme of [53], we now obtain a matrix

$$\bar{\boldsymbol{S}}^{(i)\prime\prime}(z) = \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \bar{\mathbf{S}}^{(i)\prime}[\tau] z^{-\tau} \; . \tag{12}$$

Note that $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)$ is not equal to $\bar{\boldsymbol{S}}^{(i)\prime}(z)$ by construction but is of lower order and therefore less computationally costly to update in the subsequent step. Applying (12) at each iteration enforces a monotonic contraction of the update space of the algorithm. We can therefore avoid truncation of $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)$ at each iteration, as its order is not increasing. As a result of this, we also limit the search space of (11), which negatively impacts the convergence speed of HRSMS, as we may not identify the same $\tau^{(i)}$ as an unrestricted version of the algorithm. However, we demonstrate in Sec. VI that this impact is typically not significant.

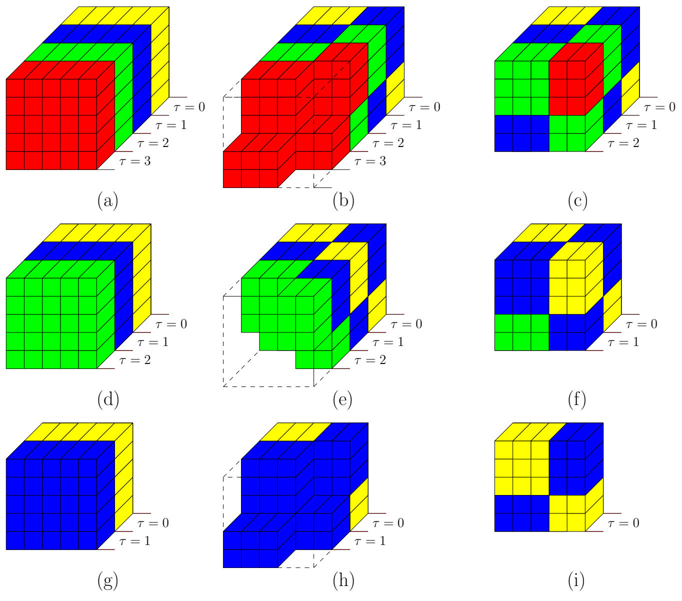


Fig. 4. (a) Matrix $\bar{\boldsymbol{S}}^{(i-1)}(z) : \mathbb{C} \to \mathbb{C}^{5\times5}$ with maximum lag $T^{(i-1)} = 3$ and $P = 2$; (b) shifting of region with maximum energy to lag zero ($\tau^{(i)} = -1$); (c) central matrix with maximum lag ($T^{(i-1)} - |\tau^{(i)}|) = 2$, $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)$, is extracted. (d) $\bar{\boldsymbol{S}}^{(i)}(z) = \mathbf{Q}^{(i)}\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i),\mathrm{H}}$; (e) shifting of region with maximum energy to lag zero ($\tau^{(i+1)} = 1$); (f) $\bar{\boldsymbol{S}}^{(i+1)\prime\prime}(z)$ extracted. (g) $\bar{\boldsymbol{S}}^{(i+1)}(z)$; (h) $\tau^{(i+2)} = -1$; (i) $\bar{\boldsymbol{S}}^{(i+2)\prime\prime}(z)$ is extracted.

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

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

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

$$\begin{aligned} \bar{\boldsymbol{S}}^{(i)}(z) &= \mathbf{Q}^{(i)}\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i),\mathrm{H}} \\ \boldsymbol{H}^{(i)}(z) &= \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z) \quad . \end{aligned} \tag{13}$$

If at this point the order of $\bar{\boldsymbol{S}}^{(i)}(z)$ is zero, we obtain a regenerated parahermitian matrix $\boldsymbol{S}^{(i)}(z) = \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\boldsymbol{H}^{(i),\mathrm{P}}(z)$, and truncate to minimise future computational complexity via $\mathbf{S}^{(i)}[\tau] \leftarrow f_{\text{trim}}(\mathbf{S}^{(i)}[\tau], \mu)$. Note that obtaining the regenerated matrix requires the use of a full-matrix representation. Following regeneration, we can continue with a half-matrix representation.

Fig. 4 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 Fig. 4(i) is equal to zero; at this point, parahermitian matrix regeneration must occur.

```
Input: $\bar{\boldsymbol{A}}(z)$, $I_D$, $P$, $\delta$, $\mu$, $\mu_t$
Output: $\bar{\boldsymbol{A}}_{11}(z)$, $\bar{\boldsymbol{A}}_{22}(z)$, $\boldsymbol{T}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise
 $\bar{\mathbf{A}}[0] \in \mathbb{C}^{M' \times M'}$
$\bar{\boldsymbol{S}}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\bar{\boldsymbol{A}}(z)\mathbf{Q}^{(0),\mathrm{H}}$, $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$, $i \leftarrow 0$,
 stop $\leftarrow 0$
do
    $i \leftarrow i+1$
    Find $\tau^{(i)}$ from (11); generate $\boldsymbol{\Lambda}^{(i)}(z)$ from (9)
    $\bar{\boldsymbol{S}}^{(i)\prime}(z) \leftarrow$
      $f_{\mathrm{shift,SMS}}(\,\bar{\boldsymbol{S}}^{(i-1)}(z), \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}\,)$
    $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
    $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z) \leftarrow \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \bar{\mathbf{S}}^{(i)\prime}[\tau]z^{-\tau}$
    $\mathbf{H}^{(i)\prime\prime}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{H}^{(i)\prime}[\tau], \mu_t)$
    Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\bar{\mathbf{S}}^{(i)\prime\prime}[0]$
    $\bar{\boldsymbol{S}}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i),\mathrm{H}}$
    $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z)$
    if $T^{(i)} = 0$ or $i > I_D$ or (14) satisfied then
        $\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\boldsymbol{H}^{(i),\mathrm{P}}(z)$
        $\mathbf{S}^{(i)}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{S}^{(i)}[\tau], \mu)$
    end
    if $i > I_D$ or (14) satisfied then
        stop $\leftarrow 1$
    end
while stop $= 0$
$\boldsymbol{T}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\bar{\boldsymbol{A}}_{11}(z)$ is top-left $(M'-P) \times (M'-P)$ block of $\bar{\boldsymbol{S}}^{(i)}(z)$
$\bar{\boldsymbol{A}}_{22}(z)$ is bottom-right $P \times P$ block of $\bar{\boldsymbol{S}}^{(i)}(z)$
```

Algorithm 3: HRSMS algorithm

After a user-defined I_D iterations, or when

$$\max_{\tau}\left\{\|\bar{\mathbf{S}}_{21}^{(I)}[\tau]\|_{\mathrm{F}}^2\ ,\ \|\bar{\mathbf{S}}_{12}^{(I)}[-\tau]\|_{\mathrm{F}}^2\right\} \leq \delta \sum_{\tau} \|\mathbf{R}[\tau]\|_{\mathrm{F}}^2 \qquad (14)$$

at some iteration I — where δ is chosen to be arbitrarily small — the HRSMS algorithm returns matrices $\bar{\boldsymbol{A}}_{11}(z)$, $\bar{\boldsymbol{A}}_{22}(z)$, and $\boldsymbol{T}(z)$. The latter is constructed from the concatenation of the elementary paraunitary matrices:

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

where $\hat{I} = \min\{I_D, I\}$. $\bar{\boldsymbol{A}}_{11}(z)$ is the top-left $(M'-P) \times (M'-P)$ block of $\bar{\boldsymbol{A}}'(z) = \boldsymbol{T}(z)\boldsymbol{R}(z)\boldsymbol{T}^{\mathrm{P}}(z)$ and $\bar{\boldsymbol{A}}_{22}(z)$ is the bottom-right $P \times P$ block of $\bar{\boldsymbol{A}}'(z)$.

The above steps of HRSMS are summarised in Algorithm 3.

C. *'Conquering' the Independent Matrices*

At this stage of PSMD, $\boldsymbol{R}(z)$ has been segmented into multiple independent parahermitian matrices, which are stored as blocks on the diagonal of $\bar{\boldsymbol{R}}'(z)$. Each matrix can now be diagonalised individually through the use of a PEVD algorithm; here, a half-matrix version [51] of the restricted update SMD algorithm from [53] is chosen, and is henceforth named half-matrix restricted update sequential matrix diagonalisation

```
Input: $\bar{\boldsymbol{B}}(z)$, $I_C$, $\epsilon$, $\mu$, $\mu_t$
Output: $\bar{\boldsymbol{C}}(z)$, $\boldsymbol{V}(z)$
Find eigenvectors $\mathbf{Q}^{(0)}$ that diagonalise $\bar{\mathbf{B}}[0] \in \mathbb{C}^{N \times N}$
$\bar{\boldsymbol{S}}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}\bar{\boldsymbol{B}}(z)\mathbf{Q}^{(0),\mathrm{H}}$, $\boldsymbol{H}^{(0)}(z) \leftarrow \mathbf{Q}^{(0)}$, $i \leftarrow 0$,
 stop $\leftarrow 0$
do
    $i \leftarrow i+1$
    Find $\{c^{(i)}, \tau^{(i)}\}$ from (16); generate $\boldsymbol{\Lambda}^{(i)}(z)$
     from (15)
    $\bar{\boldsymbol{S}}^{(i)\prime}(z) \leftarrow$
      $f_{\mathrm{shift,SMD}}(\,\bar{\boldsymbol{S}}^{(i-1)}(z), c^{(i)}, \tau^{(i)}, \boldsymbol{\Lambda}^{(i)}(z), T^{(i-1)}\,)$
    $\boldsymbol{H}^{(i)\prime}(z) \leftarrow \boldsymbol{\Lambda}^{(i)}(z)\boldsymbol{H}^{(i-1)}(z)$
    $\bar{\boldsymbol{S}}^{(i)\prime\prime}(z) \leftarrow \sum_{\tau=0}^{T^{(i-1)}-|\tau^{(i)}|} \bar{\mathbf{S}}^{(i)\prime}[\tau]z^{-\tau}$
    $\mathbf{H}^{(i)\prime\prime}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{H}^{(i)\prime}[\tau], \mu_t)$
    Find eigenvectors $\mathbf{Q}^{(i)}$ that diagonalise $\bar{\mathbf{S}}^{(i)\prime\prime}[0]$
    $\bar{\boldsymbol{S}}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\bar{\boldsymbol{S}}^{(i)\prime\prime}(z)\mathbf{Q}^{(i),\mathrm{H}}$
    $\boldsymbol{H}^{(i)}(z) \leftarrow \mathbf{Q}^{(i)}\boldsymbol{H}^{(i)\prime\prime}(z)$
    if $T^{(i)} = 0$ or $i > I_C$ or (18) satisfied then
        $\boldsymbol{S}^{(i)}(z) \leftarrow \boldsymbol{H}^{(i)}(z)\boldsymbol{R}(z)\boldsymbol{H}^{(i),\mathrm{P}}(z)$
        $\mathbf{S}^{(i)}[\tau] \leftarrow f_{\mathrm{trim}}(\mathbf{S}^{(i)}[\tau], \mu)$
    end
    if $i > I_C$ or (18) satisfied then
        stop $\leftarrow 1$;
    end
while stop $= 0$
$\boldsymbol{V}(z) \leftarrow \boldsymbol{H}^{(i)}(z)$
$\bar{\boldsymbol{C}}(z) \leftarrow \bar{\boldsymbol{S}}^{(i)}(z)$
```

Algorithm 4: HRSMD algorithm

(HRSMD). Each instance of HRSMD is provided with a parameter I_C — which defines the maximum possible number of algorithm iterations — a stopping threshold ϵ, and truncation parameters μ and μ_t. Upon completion, the HRSMD algorithm returns matrices $\boldsymbol{V}(z)$ and $\bar{\boldsymbol{C}}(z)$, which contain the polynomial eigenvectors and eigenvalues for input matrix $\bar{\boldsymbol{B}}(z)$, respectively, such that $\boldsymbol{C}(z) = \boldsymbol{V}(z)\boldsymbol{B}(z)\boldsymbol{V}^{\mathrm{P}}(z)$.

The HRSMD algorithm approximates the PEVD using a series of elementary paraunitary operations to iteratively diagonalise a parahermitian matrix $\bar{\boldsymbol{B}}(z) : \mathbb{C} \rightarrow \mathbb{C}^{N \times N}$. Each elementary paraunitary operation consists of two steps: first a delay step is used to move the column or row with the largest energy in its off-diagonal elements to lag zero; then an EVD diagonalises the lag zero matrix, transferring the shifted energy onto the diagonal.

The HRSMD algorithm is functionally very similar to HRSMS, and also employs the restricted update approach of [53]. HRSMS zeroes off-diagonal elements in order to create a block-diagonal matrix, whereas HRSMD zeroes off-diagonal elements in order to create a diagonal matrix. We therefore describe only the differences between HRSMD and HRSMS, and provide pseudocode in Algorithm 4.

In HRSMD, the product in (8) uses a paraunitary delay matrix

$$\boldsymbol{\Lambda}^{(i)}(z) = \mathrm{diag}\{\underbrace{1\ \ldots\ 1}_{c^{(i)}-1}\ z^{-\tau^{(i)}}\ \underbrace{1\ \ldots\ 1}_{N-c^{(i)}}\} \quad , \qquad (15)$$

```
Input: $\bar{\boldsymbol{S}}(z)$, $c$, $\tau_s$, $\boldsymbol{\Lambda}(z)$, $T$
Output: $\bar{\boldsymbol{S}}'(z)$
$\boldsymbol{\Gamma}(z) \leftarrow \begin{bmatrix} \gamma_{1,1}(z) & \cdots & \gamma_{1,N}(z) \\ \vdots & \ddots & \vdots \\ \gamma_{N,1}(z) & \cdots & \gamma_{N,N}(z) \end{bmatrix}$
if $\tau_s > 0$ then
    $\boldsymbol{L}(z) \leftarrow \bar{\boldsymbol{S}}(z)\boldsymbol{\Lambda}^{\mathrm{P}}(z)$
    $\gamma_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=-\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau]z^{-\tau}, & k=c,\, m \neq c \\ 0, & \text{otherwise} \end{cases}$
    $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{\tau_s}\boldsymbol{\Gamma}^{\mathrm{P}}(z)$
    $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\boldsymbol{L}(z)$
else if $\tau_s < 0$ then
    $\boldsymbol{L}(z) \leftarrow \boldsymbol{\Lambda}(z)\bar{\boldsymbol{S}}(z)$
    $\gamma_{m,k}(z) \leftarrow \begin{cases} \sum_{\tau=\tau_s+1}^{0} \mathbf{L}_{m,k}[\tau]z^{-\tau}, & m=c,\, k \neq c \\ 0, & \text{otherwise} \end{cases}$
    $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z) + z^{-\tau_s}\boldsymbol{\Gamma}^{\mathrm{P}}(z)$
    $\boldsymbol{L}(z) \leftarrow \boldsymbol{L}(z)\boldsymbol{\Lambda}^{\mathrm{P}}(z)$
else
    $\boldsymbol{L}(z) \leftarrow \bar{\boldsymbol{S}}(z)$
end
$\bar{\boldsymbol{S}}'(z) \leftarrow \sum_{\tau=0}^{T+|\tau_s|} \mathbf{L}[\tau]z^{-\tau}$
```

Algorithm 5: $f_{\text{shift,SMD}}(\cdot)$ function

which is selected based on the position of the dominant off-diagonal column or row in $\bar{\boldsymbol{S}}^{(i-1)}(z)$ •—○ $\bar{\mathbf{S}}^{(i-1)}[\tau]$, as identified by the parameter set

$$\{c^{(i)}, \tau^{(i)}\} = \arg\max_{k,\tau} \left\{ \|\hat{\mathbf{s}}_k^{(i-1)}[\tau]\|_2, \|\hat{\mathbf{s}}_{(\mathrm{r}),k}^{(i-1)}[-\tau]\|_2 \right\}, \quad (16)$$

where

$$\|\hat{\mathbf{s}}_k^{(i-1)}[\tau]\|_2 = \sqrt{\textstyle\sum_{m=1,m\neq k}^{N} |\bar{\mathbf{S}}_{m,k}^{(i-1)}[\tau]|^2} \;,$$
$$\|\hat{\mathbf{s}}_{(\mathrm{r}),k}^{(i-1)}[\tau]\|_2 = \sqrt{\textstyle\sum_{m=1,m\neq k}^{N} |\bar{\mathbf{S}}_{k,m}^{(i-1)}[\tau]|^2} \;. \quad (17)$$

A function $f_{\text{shift,SMD}}(\cdot)$ — described in Algorithm 5 — is used instead of $f_{\text{shift,SMS}}(\cdot)$. According to the $f_{\text{shift,SMD}}(\cdot)$ function: if (16) returns $\tau^{(i)} > 0$, then the $c^{(i)}$th column of $\bar{\boldsymbol{S}}^{(i-1)}(z)$ is to be shifted by $\tau^{(i)}$ lags towards lag zero. If $\tau^{(i)} < 0$, it is the $c^{(i)}$th row that requires shifting by $-\tau^{(i)}$ lags towards lag zero. Elements that are shifted beyond lag zero have to be stored as their parahermitian conjugate and appended onto the $c^{(i)}$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 $f_{\text{shift,SMD}}(\cdot)$ function shifts the polynomial in the $c^{(i)}$th position along the diagonal in opposite directions, such that this polynomial remains unaffected. An efficient implementation of $f_{\text{shift,SMD}}(\cdot)$ can therefore exclude this element from shifting operations.

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

$$\max_{k,\tau} \left\{ \|\hat{\mathbf{s}}_k^{(I)}[\tau]\|_2 \;,\; \|\hat{\mathbf{s}}_{(\mathrm{r}),k}^{(I)}[-\tau]\|_2 \right\} \leq \epsilon \quad , \quad (18)$$

where the value of ϵ is chosen to be arbitrarily small. On completion, HRSMD returns matrices $\boldsymbol{V}(z)$ and $\bar{\boldsymbol{C}}(z)$, where $\boldsymbol{C}(z) = \boldsymbol{V}(z)\boldsymbol{B}(z)\boldsymbol{V}^{\mathrm{P}}(z)$. The former is constructed from the concatenation of the elementary paraunitary matrices:

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

where $\hat{I} = \min\{I_C, I\}$.

D. Algorithm Convergence

Various members of the SMD family of algorithms have been explicitly proven to converge in [30], [31], [56], and are guaranteed to reduce a norm over all off-diagonal elements of a parahermitian matrix below any arbitrarily small value, given a sufficient number of iterations. The algorithm family includes search space strategies that limit the temporal and/or spatial application of a paraunitary similarity transform [50], [56], which are similar to the spatial restrictions applied within the HRSMS 'divide' algorithm, and the HRSMD algorithm performing the 'conquer' step. Thus the PSMD algorithm only applies numerical efficiencies to these existing SMD family members, and provided that truncation errors are sufficiently low to not substantially alter the matrix factors, the PSMD algorithm's convergence is covered by these existing proofs and formally summarised in [57], which is omitted here.

V. Convergence Metrics

If HRSMS is not executed with a sufficient number of iterations I_D or the threshold δ is too high, the generated parahermitian matrix is not perfectly block diagonal, with the matrices $\bar{\boldsymbol{A}}_{21}(z)$ and $\bar{\boldsymbol{A}}_{12}(z)$ containing non-zero energy. Discarding the latter matrices upon completion of an instance of HRSMS introduces errors that degrade the approximation given by (1). Counteracting this by an increase in I_D or decrease of δ will reduce the speed and increase the complexity of the 'divide' step and therefore the overall algorithm.

Further, higher truncation thresholds for the parahermitian and paraunitary matrices worsen the approximation given by (1) and weaken the paraunitary property of eigenvectors $\boldsymbol{F}(z)$; i.e., equality in (2) is no longer guaranteed if truncation is employed during generation of $\boldsymbol{F}(z)$. We will define metrics for these errors below.

1) Normalised Off-Diagonal Energy : Since iterative PEVD algorithms progressively minimise off-diagonal energy, a suitable metric $E_{\text{norm}}^{(i)}$, defined in [30], can be used to measure their performance; this metric normalises the off-diagonal energy in the parahermitian matrix at the ith iteration— equivalent to the square of (17)— by the total energy, which remains invariant under paraunitary operations. Computation of $E_{\text{norm}}^{(i)}$ generates squared covariance terms; therefore a logarithmic notation of $5\log_{10} E_{\text{norm}}^{(i)}$ is employed.

2) Eigenvalue Resolution: We define eigenvalue resolution as the normalised mean-squared error between the spectrally majorised ground-truth and measured eigenvalue PSDs:

$$\lambda_{\text{res}} = \frac{1}{MK}\sum_{m=1}^{M}\sum_{k=0}^{K-1}\frac{|\mathbf{D}_{m,m}[k]-\hat{\mathbf{W}}_{m,m}[k]|}{\hat{\mathbf{W}}_{m,m}[k]} , \tag{19}$$

where $\mathbf{D}[k]$ is obtained from the K-point DFT of $\mathbf{D}[\tau]$ and $\hat{\mathbf{W}}[k]$ is found by spectrally majorising the K-point DFT of ground-truth eigenvalues $\mathbf{W}[\tau]$. 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 (19) will give emphasis to the correct extraction of small eigenvalues in the presence of stronger ones, similar to the coding gain metric in [26].

3) Decomposition Mean Squared Error: Denote the mean squared reconstruction error for an approximate PEVD as

$$\text{MSE} = \frac{1}{M^2L'}\sum\nolimits_{\tau}\|\mathbf{E}_R[\tau]\|_{\text{F}}^2 , \tag{20}$$

where $\mathbf{E}_R[\tau] = \hat{\mathbf{R}}[\tau] - \mathbf{R}[\tau]$, $\hat{\boldsymbol{R}}(z) = \boldsymbol{F}^{\text{P}}(z)\boldsymbol{D}(z)\boldsymbol{F}(z)$, and L' is the length of $\boldsymbol{E}_R(z)$.

4) Paraunitarity Error: Define the paraunitarity error as

$$\eta = \frac{1}{M}\sum\nolimits_{\tau}\|\mathbf{E}_F[\tau]-\mathbf{I}_{\text{M}}[\tau]\|_{\text{F}}^2 , \tag{21}$$

where $\boldsymbol{E}_F(z) = \boldsymbol{F}(z)\boldsymbol{F}^{\text{P}}(z)$, $\mathbf{I}_{\text{M}}[0]$ 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.

5) Paraunitary Filter Length: The output paraunitary matrix $\boldsymbol{F}(z)$ can be implemented as a lossless bank of finite impulse response filters in applications; a useful metric for gauging the implementation cost of this matrix therefore is its length, L_F.

VI. Simulations and Results

A. Simulation Scenario

The simulations below have been performed over an ensemble of 10^3 instantiations of $\boldsymbol{R}(z) : \mathbb{C}\to\mathbb{C}^{M\times M}$, $M = 30$, based on the randomised source model in [30]. This source model generates $\boldsymbol{R}(z) = \boldsymbol{X}^{\text{P}}(z)\boldsymbol{W}(z)\boldsymbol{X}(z)$, whereby the diagonal $\boldsymbol{W}(z) : \mathbb{C}\to\mathbb{C}^{M\times M}$ contains the PSDs of 30 independent sources. These sources are spectrally shaped by innovation filters such that $\boldsymbol{W}(z)$ has an order of 118, and limits the dynamic range of the PSDs to about 30 dB. Random paraunitary matrices $\boldsymbol{X}(z) : \mathbb{C}\to\mathbb{C}^{M\times M}$ of order 60 perform a convolutive mixing of these sources, such that $\boldsymbol{R}(z)$ has an order of 238.

The performances of the existing SBR2 [5], SMD [30], and DCSMD [52] PEVD algorithms are compared with a newly developed half-matrix DCSMD (HDCSMD) algorithm and the proposed PSMD algorithm.

SBR2 and SMD are allowed to run for 1800 and 1400 iterations, respectively, with truncation parameters $\mu_{\text{SBR2}} = \mu_{\text{SMD}} = 10^{-6}$. DCSMD, HDCSMD, and PSMD are provided with parameters $\mu = \mu_t = \mu_s = 10^{-12}$, $\delta = 0$, $\epsilon = 0$, $I_D = 100$, $I_C = 200$, $\hat{M} = 8$, and $P = 8$. Two variants of PSMD are also tested: PSMD^1 is supplied with $\mu = \mu_t = \mu_s = 10^{-6}$ and PSMD^2 is supplied with $I_D = 400$, while all other parameters are kept the same.

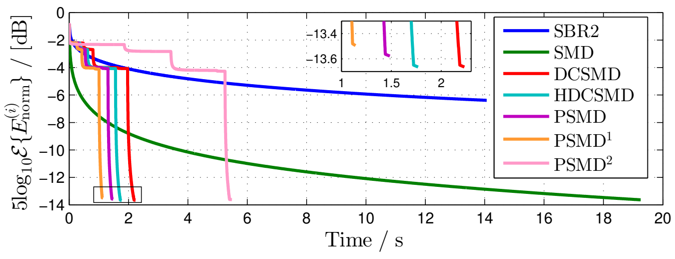


Fig. 5. Performance of PSMD, PSMD^1, and PSMD^2 relative to SBR2 [5], SMD [30], DCSMD [52], and half-matrix DCSMD for the decomposition of a 30×30 parahermitian matrix.

Simulations were 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.

B. Diagonalisation Speed

1) Without Parallelisation: The ensemble-averaged diagonalisation metric of Sec. V-1 for each of the tested PEVD algorithms is plotted against the ensemble-averaged elapsed system time at each iteration in Fig. 5. The curves demonstrate that PSMD 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 diagonalisation performance similar to the other algorithms. By utilising a restricted update approach, PSMD has sacrificed a small amount of diagonalisation performance to decrease algorithm run-time versus the otherwise functionally identical HDCSMD algorithm. Increased levels of truncation within PSMD^1 have decreased algorithm run-time but have also decreased diagonalisation performance slightly. The increase in I_D within PSMD^2 has increased the run-time of the ‘divide’ step and marginally improved diagonalisation.

The ‘stepped’ characteristics of the curves for the DaC strategies of DCSMD, HDCSMD, 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.

2) With Parallelisation: From Fig. 5, the average run-time for the PSMD algorithm for the given simulation scenario is 1.485 seconds. If the MATLAB® Parallel Computing Toolbox™ is used to parallelise the ‘conquer’ step of PSMD by spreading four instances of HRSMD across the four cores present on the simulation platform, the average run-time can be reduced to 1.075 seconds. The performance of PSMD is otherwise identical.

In this case, the use of parallelisation has dramatically reduced the run-time of the ‘conquer’ step to the point where it is negligible when compared with the run-time of the ‘divide’ step. Unfortunately, as the ‘divide’ step has to process matrices of larger spatial dimensions, it tends to be slower, and ultimately provides a relatively high lower bound for the overall run-time.

TABLE I
AVERAGE λ_{res}, MSE, η, AND L_F COMPARISON.

Method	λ_{res}	MSE	η	L_F
SBR2 [5]	1.1305	1.293×10^{-6}	2.448×10^{-8}	133.8
SMD [30]	0.0773	3.514×10^{-6}	6.579×10^{-8}	165.5
DCSMD [52]	0.0644	6.785×10^{-6}	1.226×10^{-14}	360.4
HDCSMD	0.0644	6.785×10^{-6}	1.226×10^{-14}	360.4
PSMD	0.0658	6.918×10^{-6}	4.401×10^{-15}	279.3
PSMD^1	0.0661	8.346×10^{-6}	1.303×10^{-8}	156.0
PSMD^2	0.0245	7.618×10^{-7}	1.307×10^{-14}	307.6

C. Eigenvalue Resolution

The ensemble-averaged eigenvalue resolution of (19) can be seen in column two of Tab. I. 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 worse diagonalisation performance of PSMD relative to DCSMD and HDCSMD 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^1 has slightly higher λ_{res} than PSMD. While PSMD^2 achieves a similar level of diagonalisation to PSMD, the additional effort contributed towards the 'divide' step has dramatically improved λ_{res}.

Experimental results for a single parahermitian matrix realisation in Fig. 6, showing ground truth versus extracted eigenvalues, exemplarily indicate that SMD prioritises resolution of eigenvalues with high power, and requires a high number of iterations to satisfactorily resolve eigenvalues with low power, while the DaC methods attempt to resolve all eigenvalues equally. This property of SMD has also been observed in [30]. For simplicity of the graphs in Fig. 6, only the strongest and weakest four of the 30 eigenvalues are shown, with the ground truth shown with dotted lines. A comparison of Fig. 6(a) and (b) indicates that SMD offers slightly better resolution of the first four eigenvalues, while Fig. 6(c) and (d) show that PSMD is more able to resolve the last four eigenvalues. More accurately resolving eigenvalues of low power may be advantageous when attempting to estimate the noise-only subspace in broadband angle of arrival scenarios, in which DaC techniques have already proved useful [22].

D. Mean Squared Error

The ensemble-averaged MSE of (20) for each algorithm forms column three of Tab. I. The DaC methods can be seen to introduce an error to the PEVD, and produce higher ensemble MSEs 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 PSMD^1 has resulted in marginally higher MSE. To reduce the MSE of DCSMD, HDCSMD, and PSMD in this scenario, I_D can be increased; however, this will reduce the speed of the algorithms, as more effort will be contributed to the 'divide' step. This can be observed in the results of PSMD^2, which offers the lowest MSE of any of the tested algorithms.

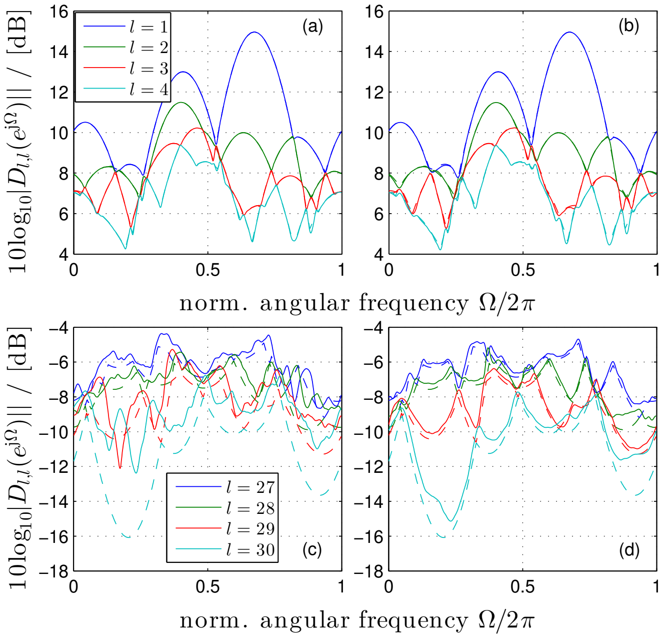


Fig. 6. Ground truth (dashed) vs extracted (solid) (a,b) strongest and (c,d) weakest four polynomial eigenvalues obtained from (a,c) SMD and (b,d) PSMD when applied to a single instance of the specified scenario.

E. Paraunitarity Error

Ensemble averages for this error defined in (21) are listed in column four of Tab. I. 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^1 has resulted in a significant increase in paraunitarity error, such that η is only slightly lower for PSMD^1 than SMD. Increasing I_D in PSMD^2 has slightly increased η, as more iterations of the 'divide' step — and therefore more truncation operations — are completed.

F. Paraunitary Filter Length

Column five of Tab. I, showing the ensemble-average paraunitary filter length of Sec. V-5, shows that DaC strategies tend to produce higher values for L_F [22], [47], [52]. However, the use of compensated row-shift truncation in PSMD has resulted in lower L_F. Using higher levels of truncation in any algorithm would reduce filter length and algorithm run-time at the expense of higher MSE and η; this relationship is observed in the results of PSMD^1, which is able to provide significantly shorter paraunitary filters than PSMD. Indeed, the filters produced are actually shorter than those given by SMD. Increasing I_D in PSMD^2 has resulted in an increase in L_F.

VII. ALGORITHM IMPLEMENTATION IN HARDWARE

Using MATLAB®'s Simulink for a graphical representation of the 'divide' and 'conquer' stages of PSMD, MATLAB®'s Embedder Coder can help to translate this modular form to C code, which can be compiled a executable binary file.

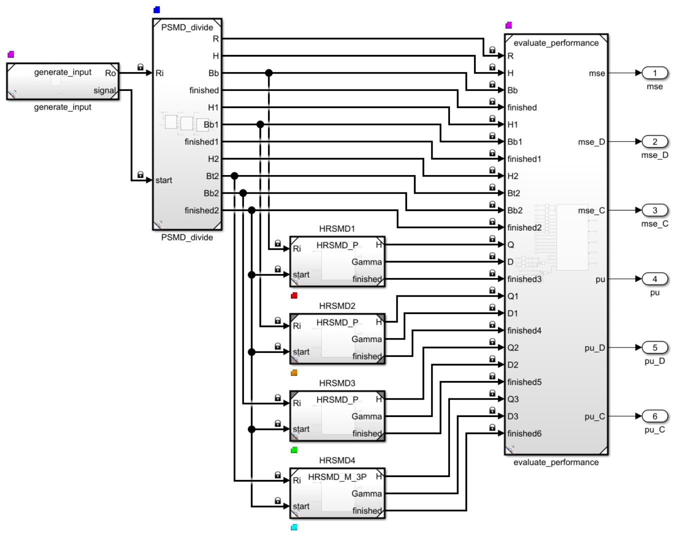


Fig. 7. Model diagram of the parallel form of the algorithm implemented in Simulink®.

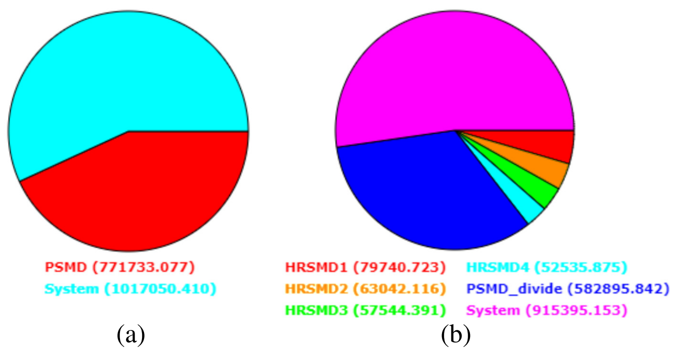


Fig. 8. Average execution time in μs for each task over 100 instances of the (a) serial and (b) parallel implementation on an Intel® i7-4700MQ CPU. ‘System’ refers to model computations not associated with the algorithm.

The PSMD implementation is detailed in Sec. VII-A, while Secs. VII-B and VII-C discuss running this on a quad core CPU and Raspberry Pi, respectively, to highlight how parallelism is exploited.

A. Modular Realisation of Algorithm

When creating a block-based modular system implementation, MATLAB®’s Simulink and Embedded Coder can generate C code, but also provide various profiling tools, and enable the concurrent execution of tasks, depending on the target hardware platform. Since PSMD utilises multiple instances of the HRSMD and HRSMS algorithms, each of these instances can be formulated as a function block and subsequently allocated to a task.

In the model of Fig. 7, a first block generates a parahermitian matrix according to Sec. VI-A. A ‘divide’ stage block hides three sequentially organised instances of HRSMS, followed by ‘conquer’ stage containing four parallel instances of HRSMD that diagonalise the four smaller parahermitian matrices. A last block provides access to result parameters for diagnostics.

If the C code generated by MATLAB®’s Embedded Coder is compiled for deployment on hardware with a compatible operating system and multiple CPU cores, each of these tasks can be managed independently and can therefore be executed concurrently. The dovetailing of tasks may however introduce some additional latency, which can affect execution time. The latter will be influenced by the slowest block, which typically is the first instance of the HRSMS ‘divide’ stage operating on an $M \times M$ matrix, while all other blocks operate on parahermitian matrices with smaller spatial dimension.

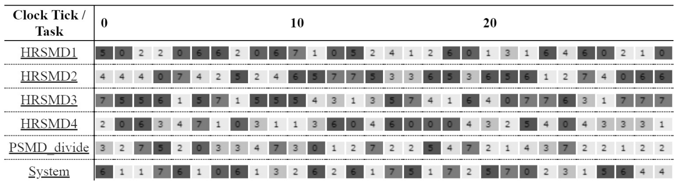
The parallel implementation of Fig. 7 can be contrasted by a serial implementation by forcing both the ‘divide’ and ‘conquer’ stages into a single block. For both serial and parallel implementations, some parameters must be preset. For memory conservation, parahermitian and paraunitary matrices are both limited to maximum lengths of 201, whereby the half-matrix method conserves half the memory space for the parahermitian matrix. To keep memory use low, we have implemented PSMD[1] from Sec. VI-A, which permitted a relatively high level of truncation and thus produced the shortest paraunitary matrices of all the considered DaC configurations.

B. Multicore CPU Implementation

For efficient implementation on an Intel® i7-4700MQ CPU, BLAS and LAPACK libraries were sourced via the Intel® Math Kernel Library. MATLAB®, and therefore the simulation results of Sec. VI implicitly rely on these to facilitate fast matrix multiplication and useful EVD algorithms. Complex double precision was used throughout, therefore numerical accuracy was equivalent to results of Tab. I.

By segmenting the serial and parallel models into tasks, a profiler native to Simulink could be utilised to evaluate the timing performance of each task individually. Average performance was ascertained by running the models over 100 instantiations, with results for the task timings shown in Fig. 8(a) and (b) for the serial and parallel versions, respectively. For the serial version, the average execution time in Fig. 8(a) is 0.772 s. For the parallel version, each task represents one of the blocks in Fig. 7. The ‘divide’ stage takes on average 0.583 s to run; thereafter, the four ‘conquer’ stages operate in parallel. With the slowest HRSMD taking an average execution time of 0.079 s, the total average execution time for the parallel implementation is 0.663 s, thus about 16% faster than the serial implementation. Also note that parallelised and compiled version, while running on the same hardware as the results provided in Fig. 5, execute globally significantly faster than the MATLAB® simulations of Sec. VI. Fig. 9 illustrates how each task is assigned a core at each sample time; the cores—four physical and four virtual ones—are numbered 0-7.

Tab. II conveys some of the resource requirements of the Intel®CPU implementation. For comparison, a DCSMD implementation is included, which uses the same serial model block layout as the one used for PSMD. For the same decomposition, it requires more time and memory than both the serial and parallel PSMD implementations, which agrees with the results obtained in Fig. 5.



Clock Tick / Task	0										10										20									
HRSMD1	5	0	2	2	0	6	6	2	0	6	7	1	0	5	2	4	1	2	6	0	1	3	1	6	4	6	0	2	1	0
HRSMD2	4	4	4	0	7	4	2	5	2	4	6	5	7	7	5	3	3	6	5	3	6	5	6	1	2	7	4	0	6	6
HRSMD3	7	5	5	6	1	5	7	1	5	5	5	4	3	1	3	5	7	4	1	6	4	0	7	7	6	3	1	7	7	7
HRSMD4	2	0	6	3	4	7	1	0	3	1	1	3	6	0	4	6	0	0	0	4	3	2	5	4	0	4	3	3	3	1
PSMD_divide	3	2	7	5	2	0	3	3	4	7	3	0	1	2	7	2	2	5	4	7	2	1	4	3	7	2	2	1	2	2
System	6	1	1	7	6	1	0	6	1	3	2	6	2	6	1	7	5	1	7	2	5	7	0	2	3	1	5	6	4	4

Fig. 9. Cores assigned to each task over the first 30 of 100 instances of the parallel implementation on an Intel® i7-4700MQ CPU.

TABLE II
AVERAGE RESOURCES UTILISED BY SERIAL AND PARALLEL IMPLEMENTATIONS OF ALGORITHM ON AN INTEL® I7-4700MQ CPU.

Model	Execution time	Memory usage	Cores used
DCSMD	1.0846 s	214709 KiB	1
PSMD, serial	0.7717 s	192883 KiB	1
PSMD, parallel	0.6626 s	192884 KiB	5

C. Raspberry Pi Implementation

To demonstrate the implementation on a hardware device that is more stand-alone than the CPU in Sec. VII-B, we have targetted a Raspberry Pi 3B+ with a 1.4 GHz 64-bit quad-core ARM processor running a Mathworks-customised Linux operating system. With the same parahermitian matrix scenario as before, the models were compiled with a Simulink support package dedicated for this hardware. Standard Linux repositories provided the BLAS and LAPACK libraries for compilation. The models were downloaded to the Raspberry Pi 3B+ in the form of '.elf' binary files; downloading as well as all other communication with the hardware platform were conducted via secure shell (SSH).

Due to unavailability of the profiling options reported in Sec. VII-A, Raspberry Pi 3B+'s CPU and memory usage were sampled with 0.1 s resolution using standard Linux commands; the results are shown for the serial model in Fig. 10. From the diagnostics of this graph, 100% of one CPU is dedicated to execution of the algorithm, while a further 100% of a second CPU is used to generate the input parahermitian matrix and log the algorithm performance. The algorithm is the longer of the two processes; we see a step from 200% to 100% when the extraneous input/output process finishes. Algorithm completion occurs when the CPU utilisation drops to 0%. We can therefore reliably measure the serial implementation by subtracting the start points from the end points. There is a baseload for the memory to hold the generated parahermitian matrix; spikes occur if processing additional resources is required.

In the parallel results of Fig. 11, we see a large initial spike to 400% CPU utilisation, since the 'divide' stage (1 core), 'conquer' stage (2 cores), and input/output process (1 core) are pipelined and all executing simultaneously. The 'conquer' stage finishes first, dropping the CPU utilisation to 200%. The path from 200% $\rightarrow$ 100% $\rightarrow$ 0% is then the same as observed for the serial implementation. The memory usage of the parallel implementation is slightly higher compared to the serial one.

The resources of Tab. III were obtained following the execution of 250 instances of the serial and parallel models on the Raspberry Pi 3B+. By exploiting the parallel implementation that the PSMD algorithm's structure affords, the execution time can be reduced by 19.55% with only a minor increase in memory usage compared to a serial PSMD realisation. Note that the core count in Tab. III excludes the input/output blocks of the model; compared to the single-core operation of the serial model, the parallel model pipelines the 'divide' with the 'conquer' stage, where the latter are executed in parallel across two cores.

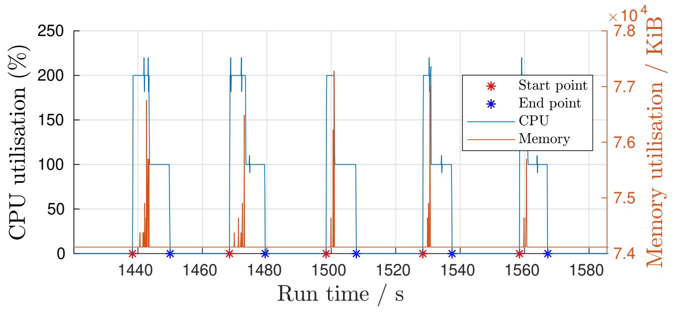


Fig. 10. CPU and memory utilisation over time of five instances of the serial model on a Raspberry Pi 3B+.

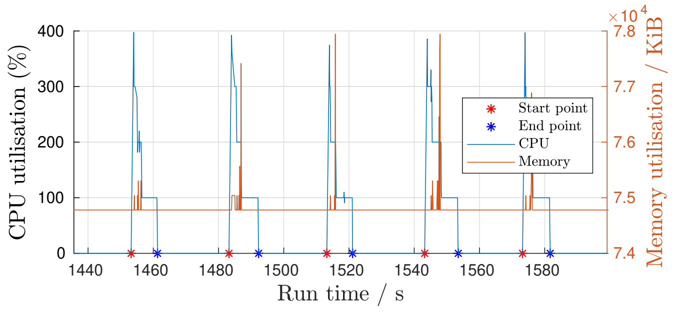


Fig. 11. CPU and memory utilisation over time of five instances of the parallel model on a Raspberry Pi 3B+.

TABLE III
AVERAGE RESOURCES UTILISED BY SERIAL AND PARALLEL IMPLEMENTATIONS OF ALGORITHM ON A RASPBERRY PI 3B+.

Model	Execution time	Memory usage	Cores used
Serial	11.647 s	74116 KiB	1
Parallel	9.370 s	74780 KiB	3

VIII. CONCLUSION

We have investigated a novel combination of — partially modified and adapted — techniques to compute the polynomial matrix EVD of a parahermitian matrix; this algorithm — named parallel-sequential matrix diagonalisation (PSMD) — makes use of a DaC approach to the PEVD, and has been shown to offer several advantages over existing algorithms. Simulation results have demonstrated that the low algorithmic complexity of the proposed method results in lower algorithm run-time than existing DaC approaches — even for non-parallel execution — with the advantage of decreasing the paraunitarity error and the paraunitary filter length. In contrast, the mean squared reconstruction error is increased slightly.

When compared with the standard iterative SBR2 and SMD algorithms, PSMD offers a significant decrease in run-time and paraunitarity error — and provides superior eigenvalue resolution — but results in higher mean squared reconstruction error and paraunitary filter length. However, the latter can be reduced at the expense of higher paraunitarity error.

While the impact of DaC algorithm parameters $\hat{M}$, P, and δ on performance metrics is not analysed here, research in [22] has investigated this topic for DCSMD and highlights the flexibility of this type of PEVD algorithm. Additional results in [47] demonstrate the increasing superiority of a DaC algorithm versus SMD when processing parahermitian matrices of increasing spatial dimension.

Two hardware implementations of the PSMD — one on an Intel® CPU, the other on a stand-alone Raspberry Pi — were demonstrated, and included the exploitation of symmetric structural features of a parahermitian matrix via the half-matrix method, and exploitation of parallelism through multi-core operation. This demonstrated enhanced execution time and memory use compared to an existing algorithm, and showed that the parallelism provides a significant improvement over a serial realisation.

When designing PEVD implementations for real applications — particularly those involving a large number of sensors — the potential for the proposed method 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.

References

[1] G. W. Stewart, “The decompositional approach to matrix computation,” *Computing in Science Eng.*, **2**(1):50–59, Jan. 2000.

[2] G. Golub and C. V. Loan, *Matrix computations*, 4th ed. Baltimore, ML: John Hopkins, 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, P. D. Baxter, T. Cooper, S. Redif, and J. Foster, “An EVD Algorithm for Para-Hermitian Polynomial Matrices,” *IEEE TSP*, **55**(5):2158–2169, May 2007.

[6] S. Icart and P. Comon, “Some properties of Laurent polynomial matrices,” in *IMA Int. Conf. Math. Signal Proc.*, Dec. 2012.

[7] P. P. Vaidyanathan, “Theory of optimal orthonormal subband coders,” *IEEE TSP*, **46**(6):1528–1543, June 1998.

[8] S. Weiss, J. Pestana, and I. K. Proudler, “On the existence and uniqueness of the eigenvalue decomposition of a parahermitian matrix,” *IEEE TSP*, **66**(10):2659–2672, May 2018.

[9] S. Weiss, J. Pestana, I. Proudler, and F. 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, Dec. 2018.

[10] C. Delaosa, F. K. Coutts, J. Pestana, and S. Weiss, “Impact of space-time covariance estimation errors on a parahermitian matrix evd,” in *IEEE SAM*, Sheffield, UK, July 2018.

[11] C. H. Ta and S. Weiss, “A design of precoding and equalisation for broadband MIMO systems,” in *Asilomar Conf. SSC*, pp. 1616–1620, Nov. 2007.

[12] R. Brandt and M. Bengtsson, “Wideband MIMO channel diagonalization in the time domain,” in *PIMRC*, pp. 1958–1962, Sept. 2011.

[13] N. Moret, A. Tonello, and S. Weiss, “MIMO precoding for filter bank modulation systems based on PSVD,” in *Proc. IEEE 73rd VTC*, pp. 1–5, May 2011.

[14] A. Sandmann, A. Ahrens, and S. Lochmann, “Resource allocation in svd-assisted optical mimo systems using polynomial matrix factorization,” in *ITG Symp. Photonic Networks*, May 2015.

[15] A. Sandmann, A. Ahrens, and S. Lochmann, “Performance analysis of polynomial matrix SVD-based broadband mimo systems,” in *SSPD*, Edinburgh, UK, Sep. 2015.

[16] A. Ahrens, A. Sandmann, E. Auer, and S. Lochmann, “Optimal power allocation in zero-forcing assisted PMSVD-based optical MIMO systems,” in *SSPD*, Edinburgh, UK, Dec. 2017.

[17] C. H. Ta and S. Weiss, “A jointly optimal precoder and block decision feedback equaliser design with low redundancy,” in *EUSIPCO*, Poznan, Poland, pp. 489–492, Sep. 2007.

[18] 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 Proc.*, **6**(7):704–712, Sep. 2012.

[19] S. Weiss, M. Alrmah, S. Lambotharan, J. McWhirter, and M. Kaveh, “Broadband angle of arrival estimation methods in a polynomial matrix decomposition framework,” in *IEEE CAMSAP*, pp. 109–112, Dec. 2013.

[20] M. Alrmah, S. Weiss, and S. Lambotharan, “An extension of the music algorithm to broadband scenarios using polynomial eigenvalue decomposition,” in *EUSIPCO*, pp. 629–633, Aug. 2011.

[21] M. Alrmah, J. Corr, A. Alzin, K. Thompson, and S. Weiss, “Polynomial subspace decomposition for broadband angle of arrival estimation,” in *SSPD*, Sep. 2014.

[22] F. K. Coutts, K. Thompson, S. Weiss, and I. Proudler, “Impact of fast-converging PEVD algorithms on broadband AoA estimation,” in *SSPD*, Dec. 2017.

[23] S. Redif, J. G. McWhirter, P. D. Baxter, and T. Cooper, “Robust broadband adaptive beamforming via polynomial eigenvalues,” in *Proc. IEEE/MTS OCEANS*, Sep. 2006.

[24] S. Weiss, S. Bendoukha, A. Alzin, F. Coutts, I. Proudler, and J. Chambers, “MVDR broadband beamforming using polynomial matrix techniques,” in *EUSIPCO*, pp. 839–843, Sep. 2015.

[25] A. Alzin, F. Coutts, J. Corr, S. Weiss, I. K. Proudler, and J. A. Chambers, “Adaptive broadband beamforming with arbitrary array geometry,” in *IET/EURASIP ISP*, Dec. 2015.

[26] S. Redif, J. McWhirter, and S. Weiss, “Design of FIR paraunitary filter banks for subband coding using a polynomial eigenvalue decomposition,” *IEEE TSP*, **59**(11):5253–5264, Nov. 2011.

[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 J. Adv. Sig. Proc.*, Mar. 2006.

[28] S. Redif, S. Weiss, and J. McWhirter, “Relevance of polynomial matrix decompositions to broadband blind signal separation,” *Sig. Proc.*, **134**:76–86, May 2017.

[29] 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 *SSPD*, London, UK, Dec. 2017.

[30] S. Redif, S. Weiss, and J. McWhirter, “Sequential matrix diagonalization algorithms for polynomial EVD of parahermitian matrices,” *IEEE TSP*, **63**(1):81–89, Jan. 2015.

[31] J. Corr, K. Thompson, S. Weiss, J. McWhirter, S. Redif, and I. Proudler, “Multiple shift maximum element sequential matrix diagonalisation for parahermitian matrices,” in *IEEE SSP*, pp. 312–315, June 2014.

[32] Z. Wang, J. G. McWhirter, J. Corr, and S. Weiss, “Multiple shift second order sequential best rotation algorithm for polynomial matrix EVD,” in *EUSIPCO*, pp. 844–848, Sep. 2015.

[33] J. Corr, K. Thompson, S. Weiss, J. G. McWhirter, and I. K. Proudler, “Causality-Constrained multiple shift sequential matrix diagonalisation for parahermitian matrices,” in *EUSIPCO*, pp. 1277–1281, Sep. 2014.

[34] A. Tkacenko and P. Vaidyanathan, “Iterative greedy algorithm for solving the fir paraunitary approximation problem,” *IEEE TSP*, **54**(1):146–160, Jan. 2006.

[35] A. Tkacenko, "Approximate eigenvalue decomposition of para-hermitian systems through successive FIR paraunitary transformations," in *IEEE ICASSP*, Dallas, TX, pp. 4074–4077, Mar. 2010.

[36] M. Tohidian, H. Amindavar, and A. M. Reza, "A DFT-based approximate eigenvalue and singular value decomposition of polynomial matrices," *EURASIP J. Adv. Sig. Proc.*, **93:**, Dec. 2013.

[37] F. K. Coutts, K. Thompson, J. Pestana, I. Proudler, and S. Weiss, "Enforcing eigenvector smoothness for a compact DFT-based polynomial eigenvalue decomposition," in *IEEE SAM*, July 2018.

[38] S. Weiss, I. K. Proudler, F. K. Coutts, and J. Pestana, "Iterative approximation of analytic eigenvalues of a parahermitian matrix EVD," in *IEEE ICASSP*, Brighton, UK, May 2019.

[39] J. G. McWhirter and Z. Wang, "A novel insight to the SBR2 algorithm for diagonalising para-hermitian matrices," in *11th IMA Conf. Maths Sig. Proc.*, Birmingham, UK, Dec. 2016.

[40] S. Kasap and S. Redif, "FPGA-based design and implementation of an approximate polynomial matrix EVD algorithm," in *2012 Int. Conf. on Field-Prog. Tech.*, pp. 135–140, Dec. 2012.

[41] ——, "FPGA implementation of a second-order convolutive blind signal separation algorithm," in *21st Sig. Proc. & Comms Apps Conf.*, Apr. 2013.

[42] ——, "Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-hermitian polynomial matrices," *IEEE TVLSI*, **22**(3):522–536, Mar. 2014.

[43] J. Foster, J. G. McWhirter, and J. Chambers, "Limiting the order of polynomial matrices within the SBR2 algorithm," in *IMA Int. Conf. Math. Signal Proc.*, Dec. 2006.

[44] C. H. Ta and S. Weiss, "Shortening the order of paraunitary matrices in SBR2 algorithm," in *Int. Conf. Inf., Comm. and Sig. Proc.*, Dec. 2007.

[45] J. Corr, K. Thompson, S. Weiss, I. Proudler, and J. McWhirter, "Row-shift corrected truncation of paraunitary matrices for PEVD algorithms," in *EUSIPCO*, pp. 849–853, Sep. 2015.

[46] ——, "Shortening of paraunitary matrices obtained by polynomial eigenvalue decomposition algorithms," in *SSPD*, Sep. 2015.

[47] F. K. Coutts, K. Thompson, S. Weiss, and I. Proudler, "Analysing the performance of divide-and-conquer sequential matrix diagonalisation for large broadband sensor arrays," in *IEEE SIPS*, Oct. 2017.

[48] J. Corr, K. Thompson, S. Weiss, J. McWhirter, and I. Proudler, "Cyclic-by-row approximation of iterative polynomial EVD algorithms," in *SSPD*, Sep. 2014.

[49] F. K. Coutts, J. Corr, K. Thompson, S. Weiss, I. Proudler, and J. McWhirter, "Complexity and search space reduction in cyclic-by-row PEVD algorithms," in *Asilomar Conf. SSC*, Nov. 2016.

[50] J. Corr, K. Thompson, S. Weiss, I. Proudler, and J. McWhirter, "Reduced search space multiple shift maximum element sequential matrix diagonalisation algorithm," in *IET/EURASIP ISP*, London, UK, Dec. 2015.

[51] F. K. Coutts, J. Corr, K. Thompson, S. Weiss, J. Proudler, and J. McWhirter, "Memory and complexity reduction in parahermitian matrix manipulations of PEVD algorithms," in *EUSIPCO*, pp. 1633–1637, Aug. 2016.

[52] F. K. Coutts, J. Corr, K. Thompson, I. Proudler, and S. Weiss, "Divide-and-conquer sequential matrix diagonalisation for parahermitian matrices," in *SSPD*, Dec. 2017.

[53] F. K. Coutts, K. Thompson, I. Proudler, and S. Weiss, "Restricted update sequential matrix diagonalisation for parahermitian matrices," in *IEEE CAMSAP*, Dec. 2017.

[54] A. Jafarian and J. McWhirter, "A novel method for multichannel spectral factorization," in *EUSIPCO*, pp. 1069–1073, Aug. 2012.

[55] J. Corr, K. Thompson, S. Weiss, I. Proudler, and J. McWhirter, "Reduced search space multiple shift maximum element sequential matrix diagonalisation algorithm," in *IET Int. Conf. ISP*, Dec. 2015.

[56] J. Corr, K. Thompson, S. Weiss, J. McWhirter, and I. Proudler, "Maximum energy sequential matrix diagonalisation for parahermitian matrices," in *48th Asilomar Conf. SSC*, pp. 470–474, Nov. 2014.

[57] F.K. Coutts, "Algorithmic enhancements to polynomial matrix factorisations," Ph.D. diss., Univ. of Strathclyde, May 2019.