Algorithms and Techniques for Polynomial Matrix Decompositions

Joanne A. Foster

A Thesis submitted for the degree of Doctor of Philosophy

School of Engineering

Cardiff University

May 2008

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

The concept of polynomial matrices is introduced and the potential application of polynomial matrix decompositions is discussed within the general context of multi-channel digital signal processing. A recently developed technique, known as the second order sequential rotation algorithm (SBR2), for performing the eigenvalue decomposition of a para-Hermitian polynomial matrix (PEVD) is presented. The potential benefit of using the SBR2 algorithm to impose strong decorrelation on the signals received by a broadband sensor array is demonstrated by means of a suitable numerical simulation. This demonstrates how the polynomial matrices produced as a result of the PEVD can be of unnecessarily high order. This is undesirable for many practical applications and slows down the iterative computational procedure.

An effective truncation technique for controlling the growth in order of these polynomial matrices is proposed. Depending on the choice of truncation parameters, it provides an excellent compromise between reduced order polynomial matrix factors and accuracy of the resulting decomposition. This is demonstrated by means of a set of numerical simulations performed by applying the modified SBR2 algorithm with a variety of truncation parameters to a representative set of test matrices.

Three new polynomial matrix decompositions are then introduced - one for implementing a polynomial matrix QR decomposition (PQRD) and two for implementing a polynomial matrix singular value decomposition (PSVD). Several variants of the PQRD algorithm (including polynomial order reduction) are proposed and compared by numerical simulation using an appropriate set of test matrices. The most effective variant w.r.t. computational speed, order of the polynomial matrix factors and accuracy of the resulting decomposition is identified.

The PSVD can be computed using either the PEVD technique, based on the SBR2 algorithm, or the new algorithm proposed for implementing the PQRD. These two approaches are also compared by means of computer simulations which demonstrate that the method based on the PQRD is numerically superior.

The potential application of the preferred PQRD and PSVD algorithms to multiple input multiple output (MIMO) communications for the purpose of counteracting both co-channel interference and inter-symbol interference (multi-channel equalisation) is demonstrated in terms of reduced bit error rate by means of representative computer simulations.

Acknowledgements

I gratefully acknowledge the help, support and invaluable advice over the last three years of both my supervisors, Prof. John McWhirter at Cardiff University and Prof. Jonathon Chambers at Loughborough University. I would also like to thank Dr. Stephan Weiss and Dr. Saeid Sanei for their constructive suggestions.

Publications

- J. Foster, J. G. McWhirter and J. Chambers, "Limiting the Order of Polynomial Matrices Within the SBR2 Algorithm," *IMA Conference Mathematics in Signal Processing, Cirencester*, 2006.

- 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(6), pp. 2158-2169, 2007.

- J. Foster, J. G. McWhirter and J. Chambers, "An Algorithm for Computing the QR Decomposition of a Polynomial Matrix," *15th International Conference on Digital Signal Processing, Cardiff*, 2007.

- J. Foster, J.G. McWhirter and J.Chambers, "A Polynomial Matrix QR Decomposition with Application to MIMO Channel Equalisation," *Proc.* 41^{st} *Asilomar Conference on Signals, Systems and Computers, Asilomar (Invited Talk)*, November, 2007.

- J. Foster, J.G. McWhirter and J.Chambers, "An Algorithm for Calculating the QR Decomposition of a Polynomial Matrix," *Manuscript in preparation.*

- J. Foster, J.G. McWhirter and J.Chambers, "An Algorithm for Calculating the Singular Value Decomposition of a Polynomial Matrix," *Manuscript in preparation.*

Contributions

- **An algorithm for calculating the QR decomposition of a polynomial matrix.** The potential application of the decomposition is to MIMO communication systems, where it is often required to reconstruct data sequences that have been distorted due to the effects of co-channel interference and mulitpath propagation leading to intersymbol interference. If the polynomial channel matrix for the system is known, then this decomposition algorithm can be used to reduce the problem of MIMO channel equalisation into an upper-triangular system of polynomial equations, which can be easily solved using back substitution and a standard equalisation scheme for a SISO problem. The capability of this application of the decomposition has been demonstrated with some simulated results.

- **An algorithm for calculating the singular value decomposition of a polynomial matrix.** Again, the capability of the decomposition to the potential application to MIMO communication systems is illustrated by some simulated results and has been shown to offer some advantages over an existing method for calculating this decomposition using the SBR2 algorithm.

- **The energy based truncation method for polynomial matrices.** This method can be used within any of the polynomial matrix decomposition algorithms to reduce the orders of the polynomial matrices and consequently the computational time taken to implement the algorithms. This is particularly important for the potential application of any of the algorithms to MIMO communication problems, where the order of the matrices is directly proportional to the computational complexity of the application.

Contents

List of Figures

List of Tables

List of Abbreviations

BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
BSS	Blind Source Separation
CCI	Co-Channel Interference
CPGR	Complete Polynomial Givens Rotation
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
ECG	Electrocardiogram
EEG	Electroencephalography
EPGR	Elementary Polynomial Givens Rotation
EPHR	Elementary Polynomial Householder Reflection
EVD	EigenValue Decomposition
FIR	Finite Impulse Response
HOS	Higer-Order Statistics
ICA	Independent Component Analysis
IIR	Infinite Impulse Response
ISI	Inter-Symbol Interference
LTI	Linear Time Invariant
MAP	Maximum a Posteriori Probability
MFD	Matrix Fraction Description
MIMO	Multi-Input Multi-Output
MLSE	Maximum Likelihood Sequence Esimator
OFDM	Orthogonal Frequency-Division Mulitplexing
PEVD	Polynomial matrix EigenValue Decomposition
PQRD	Polynomial matrix QR Decomposition
PQRD-BS	PQRD By Steps algorithm
PQRD-BC	PQRD By Columns algorithm
PQRD-SBR	PQRD By Sequential Best Rotation algorithm
PSD	Power Spectral Density
PSVD	Polynomial matrix Singular Value Decomposition
QPSK	Quadrature Phase Shift Keying
QRD	QR Decomposition
RSNR	Signal-to-Noise Ratio at the Receiver
SBR2	Sequential Best Rotation algorithm
SBR	Sequential Best Rotation

SISO	Single-Input Single-Output
SNR	Signal-to-Noise Ratio
SOS	Second-Order Statistics
SVD	Singular Value Decomposition
WSS	Wide Sense Stationary
ZLS	Zero-Lag Specification
ZLSS	Zero-Lag Specification Step

List of Notation

$E[\cdot]$	Expection operator
$\mathbf{I}_n$	$n \times n$ identity matrix
$\mathbf{0}_{p\times q}$	$p \times q$ matrix of zeros
$\underline{\mathbf{A}}(z)$	Polynomial matrix
$\det(\mathbf{A})$	Determinant of $\mathbf{A}$
$\|\cdot\|_F$	Frobenius norm
$p(\cdot)$	Probability densitiy
$\widetilde{(\cdot)}$	Paraconjugation
$[\underline{\mathbf{A}}(z)]_{jkt}$	The coefficient of z^{-t} in the $(j,k)^{th}$ element of $\underline{\mathbf{A}}(z)$
$a_{jk}(t)$	The coefficient of z^{-t} in the $(j,k)^{th}$ element of $\underline{\mathbf{A}}(z)$
$\underline{\mathbb{C}}^{a\times b}$	The set of polynomial matrices with complex coefficients with a rows and b columns
$\mathbb{C}^{a\times b\times c}$	The set of polynomial matrices of order c with complex coefficients with a rows and b columns
$(\cdot)_*$	Complex conjugate
$(\cdot)^H$	Hermitian conjugate of a matrix or vector
$(\cdot)^T$	Transposition of a matrix or vector
$tr(\mathbf{A})$	Trace of the matrix $\mathbf{A}$

Chapter 1

Introduction

1.1 Motivation

Digital signal processing (DSP) became a major area of interest in the mid 1960s when high speed digital computers became readily available for research [1]. From the advances in technology built upon this research, many potential applications were realised and a greater need for the advancement of DSP techniques was identified over the ensuing decades. The emergence of the Internet and major developments in wireless technology and mobile telecommunications have in particular been underpinned by DSP. As an example of the commercial return of research in DSP, wireless revenues are currently growing between 20% and 30% per year and are likely to continue this trend in the forseeable future [2].

Research in digital communications can be divided into many subsections, which include the detection and estimation of signals that have been transmitted over a channel. The main aim of these areas is to obtain an estimate of the transmitted signal from the received signals in an efficient and robust way. The term blind source separation (BSS) is used to describe the process of recovering a set of source signals from a collection of observed mixtures of these sources, when both the source signals and the mixing model are unknown. It may be desirable to recover all sources from the recorded mixtures or at the very least isolate a particular source. Alternatively, it can often be useful to establish information about how the source signals have been mixed and therefore gain some understanding about the physical mixing process observed.

The solution of source separation problems has been the focus of much research over the last couple of decades and the problem can generally be divided into two categories, depending on whether the signals have been instantaneously or convolutively mixed. In the instantaneous case, the relative delay can be modelled as a simple phase shift. Therefore, the sensors receive the same time sample of the mixed source signals and so the mixing matrix required to describe this scenario has complex scalar entries. In the more complicated yet more realistic convolutive case, the set of source signals are received at an array of sensors over multiple paths and with different time delays. The multiple paths arise from scattering, reflection and diffraction of the signals in the channel [2]. Furthermore, the finite propagation speed will also influence the signals over their transmission, which will also be typically corrupted by noise. This more complicating scenario is referred to as convolutive mixing and each element of the mixing matrix required to describe this situation will be a finite impulse response (FIR) filter. This FIR filter will take the form of a polynomial in the indeterminate variable z^{-1}, which is used to represent a unit delay.

Initially, most of the research in the field of BSS [3, 4] was concentrated on the simpler case of narrowband signals, where instantaneous mixing takes place. Many effective algorithms have been developed to solve the problem and have been applied to a wide range of useful applications. Within this problem elements of numerical linear algebra, such as matrix decomposition methods, have proven to be a useful tool for simplifying the problem [5]. For example, both the eigenvalue and singular value decompositions (EVD and SVD) can be used as a preprocessing step to many instantaneous BSS algorithms, where decorrelation of the received signals is required. These scalar matrix decompositions can also be used to simplify multivariate analytical problems and can therefore be applied to a diverse range of problems. Consequently, the EVD and SVD can be used to reduce the number of computations required for many operations, thus allowing computational convenience, and ensure numerical robustness [6].

In the past decade much BSS research has been focused on the separation of convolutive mixtures, where each element of the mixing matrix is a polynomial with an associated set of coefficients. Most techniques to solve this problem transform the signals to the frequency

domain, separating them into a number of narrowband problems, where an instantaneous but complex BSS algorithm can be applied. Other techniques operate entirely in the time domain, where as with the instantaneous methods, a preprocessing step would be of use. Most notably, an algorithm suitable for calculating the EVD of a para-Hermitian polynomial matrix has been developed in [7]. This algorithm is called the sequential best rotation algorithm (SBR2) and constitutes a direct extension of Jacobi's EVD algorithm from scalar to polynomial matrices. The algorithm was developed as part of a two-stage convolutive BSS algorithm, as it has the capability of performing strong decorrelation on a set of convolutively mixed signals [8]. The algorithm has also been applied to problems concerning multichannel data compression [9, 10] and more recently it has been used as a technique for designing orthogonal space-time channels for optimal data transmission [11–15]. This list does not include all applications of the algorithm to broadband signal processing, which will be as diverse as the applications of the conventional EVD for scalar matrices to narrowband signal processing.

The motivation behind this thesis is the development of other polynomial matrix decompositions using the methodology introduced for the SBR2 algorithm. In particular, algorithms for calculating the QR decomposition and SVD of a polynomial matrix are developed. These algorithms have been proven to converge at least as well as the SBR2 algorithm and are also numerically stable. Note that these decompositions of a polynomial matrix cannot be calculated using the conventional techniques for formulating the same decomposition of a scalar matrix, as each element is now a polynomial with an associated set of coefficients. The applications of these decomposition techniques to broadband signal processing are analogous to the applications of the scalar matrix equivalent decomposition to narrowband problems. In particular, the application of the decompositions to multi-input multi-output (MIMO) communication systems is examined and some results based on numerical simulations presented to support the potential applications. In this case, the polynomial matrix decomposition techniques can be used to simplify the problem and ease computation in a similar way to how scalar matrix decompositions could be used to solve a set of linear equations.

1.2 Problem Statement and Aims of Thesis

The SBR2 algorithm is a technique that has been developed for calculating the eigenvalue decomposition of a para-Hermitian polynomial matrix [7]. It was developed as part of an algorithm for solving the convolutive BSS problem, and can be used to enforce strong decorrelation upon a set of convolutively mixed signals, by diagonalising the polynomial space-time covariance matrix of the mixed signals. This algorithm provides the foundation for the research presented in this thesis. The main aims of this thesis are now described below:

1. To decrease the computational load within the SBR2 algorithm by introducing an efficient polynomial matrix truncation method, which can be used to significantly reduce the order of the polynomial matrices within the algorithm, whilst not compromising the accuracy of the decomposition performed. Without implementing some type of truncation method the order of the polynomial matrices within the SBR2 algorithm can become unnecessarily large. A couple of techniques to tackle this problem are proposed and the level of decomposition is assessed when using each of these techniques. The large orders of the polynomial matrices is a particular problem for the application of this algorithm to MIMO channel equalisation problems, where it is used to calculate the SVD of a polynomial channel matrix. If the orders of the polynomial matrices obtained by the decomposition are too large then the method becomes computationally too complex to utilise. Extensive examples have shown that even for a channel matrix of a relatively small order with coefficients having a constant power profile as observed in wireless communications, the orders of the generated polynomial matrices are too large to use easily for this application. Note that for this application, the SBR2 algorithm does not directly operate on the polynomial channel matrix.

2. Secondly, the thesis aims to introduce an efficient algorithm for calculating the QR decomposition of a polynomial matrix. The algorithm operates by applying the same elementary paraunitary operations as used in the SBR2 algorithm. The potential application of this decomposition is also to MIMO communication systems, where channel equalisation is required. This decomposition has the considerable numerical advantage

that it operates on the polynomial channel matrix directly and is found to generally yield polynomial matrices of a smaller order than those obtained by the SBR2 algorithm.

3. An algorithm for computing the SVD of a polynomial matrix is introduced and a proof of convergence for this algorithm is also presented. This new algorithm operates by iteratively calculating the QR decomposition of a polynomial matrix. This algorithm can also be used within MIMO channel equalisation problems, as an alternative to the SBR2 algorithm. As this algorithm also directly operates on the polynomial channel matrix, it will also typically generate polynomial matrices of a smaller order than those obtained using the SBR2 algorithm to formulate the SVD of a polynomial matrix.

4. Finally, this thesis examines the potential applications of the two polynomial matrix decompositions, i.e. the polynomial matrix QR decomposition (PQRD) and polynomial matrix SVD (PSVD), which have been introduced in this thesis. Application motivated examples are given to illustrate their relevance and capabilities to the specified applications.

1.3 Organisation of Thesis

The second chapter introduces the concept of blind source separation and provides a review of current methods for solving this problem for both the instantaneous and convolutive cases. This chapter includes a discussion on the background of polynomial matrices and in particular describes how they arise in signal processing. Finally, at the end of this chapter, existing techniques for calculating polynomial matrix decompositions are reviewed.

The third chapter of this thesis introduces the SBR2 algorithm, which can be used to calculate the EVD of a para-Hermitian polynomial matrix and operates by applying a series of elementary paraunitary matrices. This algorithm can be used as a time-domain approach for achieving strong decorrelation of a set of convolutively mixed signals. This application of the algorithm is discussed and an example given to illustrate how the algorithm operates. This example illustrates the one problem with this algorithm, which is the growing orders of

the polynomial matrices. This algorithm can also be used to obtain the SVD of a polynomial matrix, however this will not be discussed in detail until the seventh chapter.

The fourth chapter introduces two truncation methods which can be applied to the polynomial matrices within the SBR2 algorithm to ensure that the orders of these matrices do not grow unnecessarily large within the algorithm. The energy based truncation method can drastically decrease the the computational load of the algorithm, whilst allowing the algorithm to obtain a highly accurate polynomial matrix decomposition. Results are given to illustrate the advantages of truncating the orders of the polynomial matrices throughout the algorithm.

The fifth chapter introduces three algorithms for calculating the QR decomposition of a polynomial matrix, these are referred to as the PQRD-BS, the PQRD-BC and the PQRD-SBR algorithms. The three algorithms employ the same elementary polynomial transformation matrices as used within the SBR2 algorithm to transform a polynomial matrix into an upper triangular polynomial matrix. Convergence of each of the three algorithms is discussed and their performance demonstrated by applying them to a simple numerical example.

The sixth chapter introduces a set of polynomial matrices, each with slightly different characteristics, which are then used for testing the three different PQRD algorithms discussed in the previous chapter. The results found when applying the algorithms to the set of test matrices illustrate the different qualities of the algorithms and exemplify any problems or disadvantages of using them. In particular, this chapter confirms that the PQRD-BC algorithm typically offers the best performance, often requiring the least number of iterations and computational time to converge. Consequently, this algorithm will generally produce an upper triangular polynomial matrix with the smallest order of the three algorithms and the decomposition performed will typically be more accurate. The polynomial matrix truncation methods introduced in the fourth chapter can also be implemented within any of the PQRD algorithms.

It can be shown that the QR decomposition of a scalar matrix can be iteratively calculated and used to obtain the SVD of a scalar matrix. Extending this idea to polynomial matrices, the seventh chapter introduces an SVD algorithm suitable for polynomial matrices. This

algorithm operates by iteratively applying the most efficient polynomial matrix QR decomposition algorithm from the fifth chapter. The algorithm has been proven to converge and shown to offer some advantages over the existing technique of calculating the SVD of a polynomial matrix using the SBR2 algorithm. The main advantage is its ability to control how small the magnitude of the off-diagonal coefficients of the transformed diagonal matrix must be driven - something that cannot be achieved with the existing approach. Consequently, a more accurate polynomial matrix decomposition can be formulated. Furthermore, to obtain the same level of decomposition (in terms of ensuring the magnitude of all off-diagonal coefficients is less than a specified value) using the two different approaches, the PQRD approach requires significantly less iterations and, as a consequence, the orders of the polynomial matrices generated by the decomposition are often shorter. The two methods are compared by means of a numerical example to demonstrate the advantages of the new PQRD approach.

The eighth chapter of the thesis briefly explains some of the potential applications of the three polynomial matrix decompositions, the polynomial matrix EVD (PEVD), the PQRD and the PSVD. The applications of the two decomposition algorithms introduced in this thesis for calculating the PQRD and the PSVD, are illustrated by applying the algorithms to some simple application motivated examples. Note that it may be beneficial for some readers to look at this chapter before reading the detailed descriptions of the algorithms presented in the previous chapters.

The final chapter concludes the research presented in the thesis and outlines how this work could be continued in the future. Appendices are then included to provide some results that have been commented on, but are not included in the main body of the text.

1.4 Notation

Matrices are denoted as upper case bold characters and vectors by lower case bold characters. The subscripts *, ^{T} and ^{H} denote complex conjugate, matrix transposition and matrix Hermitian conjugate respectively. A $p \times p$ identity matrix will be denoted as $\mathbf{I}_p$ and a $p \times q$ matrix with zero entries will be referred to as $\mathbf{0}_{p\times q}$. Let $\mathbb{C}$ and $\mathbb{R}$ denote the field of complex

numbers and the field of real numbers respectively.

The underline notation, used with a matrix, vector or scalar, is used to denote a polynomial to avoid confusion with the notation used for the z-transform of a variable. For example, $\underline{\mathbf{A}}(z)$ will denote a polynomial matrix in the indeterminate variable z^{-1}. Let $\underline{\mathbf{a}}_{ij}(z)$ denote the $(i,j)^{th}$ polynomial element of the matrix $\underline{\mathbf{A}}(z)$. The coefficient associated with the $(i,j)^{th}$ polynomial element of $\underline{\mathbf{A}}(z)$, corresponding to a delay of z^{-t}, will be denoted as $a_{ij}(t)$ or occasionally, if this notation is not suitable, it is denoted as $[\underline{\mathbf{A}}(z)]_{jkt}$. Let the set of polynomial matrices, with complex coefficients, be denoted by $\underline{\mathbb{C}}^{a\times b}$ where a denotes the number of rows and b the number of columns of the polynomial matrix. If the order of the polynomial matrix is also known, for example suppose it is c, then alternatively the set could be denoted by $\mathbb{C}^{a\times b\times c}$. Similarly, if the polynomial matrix has real coefficients then the set of polynomial matrices with a rows and b columns, but an unspecified order, is denoted as $\underline{\mathbb{R}}^{a\times b}$. If the order of the matrix is known and is c, then the set is denoted as $\mathbb{R}^{a\times b\times c}$.

The tilde notation, $(\widetilde{\cdot})$, used above a polynomial matrix is used to denote the paraconjugate. Finally $\|\cdot\|_F$ is used to denote the Frobenius norm of a matrix and will also be referred to as the F-norm.

Chapter 2

Background to Convolutive Mixtures and Polynomial Matrices

2.1 Introduction

The problem of source separation can be considered in different ways depending on how the sources have been mixed. In the instantaneous case, a matrix of complex scalars is sufficient to describe the mixing. Clearly, this model is not realistic for many applications, as signals can often take multiple paths with different time delays. In this situation a matrix of polynomial elements is commonly required to describe the mixing process.

This chapter firstly discusses the simpler instantaneous mixing model. Methods for achieving instantaneous blind source separation are briefly discussed and, in particular, the role of scalar matrix decompositions to this problem and other potential applications of these decompositions is examined. The chapter then discusses the more complicated scenario where convolutive mixtures arise. In this situation, the channel matrix required to express the mixing takes the form of a polynomial matrix, where each element is a finite impulse response (FIR) filter. Polynomial matrices have therefore been used extensively in recent years in the area of digital signal processing, but they can also be used to describe the multivariable transfer function associated with a multi-input multi-output (MIMO) communication system. Other examples of their applications include broadband adaptive sensor array processing, broadband subspace decomposition and also digital filter banks for subband coding or data compression [10, 16, 17].

The topic of this thesis is the development of algorithms for the computation of polynomial matrix decompositions. There are already existing decomposition techniques, which operate on polynomial matrices. These include routines such as the Smith-McMillan decomposition for transforming a polynomial matrix into a diagonal polynomial matrix [16, 18, 19] and a method introduced by Vaidyanathan for factorising a paraunitary polynomial matrix [16]. However, little research has focused on extending standard scalar matrix procedures for calculating decompositions such as the eigenvalue decomposition (EVD), singular value decomposition (SVD) or QR decomposition (QRD) to polynomial matrices. The first, of two existing techniques, is an EVD routine for polynomial matrices developed by Lambert [20]. However, this routine operates by converting the polynomial matrices into the frequency domain and therefore offers only an approximate decomposition. A very different approach has been used to develop the SBR2 algorithm [7, 8, 21], which is an alternative EVD routine for polynomial matrices. This algorithm constitutes a natural generalisation of Jacobi's algorithm from scalar matrices to polynomial matrices. The majority of these polynomial matrix decomposition techniques are discussed in more detail at the end of this chapter. However, the SBR2 algorithm is discussed in the following chapter, as this algorithm forms the basis of the work presented in this thesis. The potential applications of polynomial matrix decompositions to broadband signal processing are examined in Chapter 8 and are often seen to analogous to the applications of the scalar matrix decompositions to narrowband signal processing.

2.2 Instantaneous Mixtures

2.2.1 The Mixing Model

In the instantaneous (narrowband) case, the propagation of the q source signals, $\mathbf{s}(t) = [s_1(t), s_2(t), \ldots, s_q(t)]^T \in \mathbb{C}^{q\times 1}$ where $t \in \{0, \ldots, T-1\}$, to an array of p sensors can be expressed as

$$\mathbf{x}(t) = \mathbf{C}\mathbf{s}(t) + \mathbf{n}(t), \tag{2.1}$$

where $\mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_p(t)]^T \in \mathbb{C}^{p\times 1}$ denotes the set of p received signals, which are each formulated as a sum of differently weighted source signals, and $\mathbf{n}(t) \in \mathbb{C}^{p\times 1}$ denotes the additive noise observed at the receiver, which is assumed to have variance $\sigma^2\mathbf{I}_p$. The matrix $\mathbf{C} \in \mathbb{C}^{p\times q}$ denotes the mixing or channel matrix for the model and has complex scalar elements c_{jk} for $j = 1, \ldots, p$ and $k = 1, \ldots, q$, which will represent the relative phase and amplitude of the k^{th} signal at the j^{th} sensor. It is generally assumed that there must be at least as many sensors as sources, i.e. $p \geq q$. Note that some work has been carried out on the underdetermined case where $p < q$ [22], however, this is not the focus of this thesis and so this case is disregarded.

2.2.2 Source Separation

The term blind source separation (BSS) is used to describe the process of recovering a set of unobserved source signals from a collection of observed mixtures, such as those demonstrated by equation (2.1), without explicit knowledge of the mixing matrix or precise signal information. If the mixing matrix of the system is known, then classical linear algebra methods can be used to determine the source signals. Even in the case where the mixing matrix is rank deficient, then the pseudo-inverse of the matrix can be calculated [6]. However, if this matrix is unknown then the problem is much more difficult and without some prior knowledge, it is impossible to uniquely determine the source signals, but they can be determined up to certain fundamental indeterminacies. Note that the solution to this problem has a diverse range of applications, for example it has been used to successfully understand biomedical signals, such as those obtained from Electrocardiogram (ECG) and Electroencephalography (EEG) readings [23, 24], within financial market analysis and even used in the design of hearing aids [25, 26].

2.2.3 Algorithms for Instantaneous BSS

One method of performing BSS, probably the most widely used, is to exploit the statistical properties of the signals and to assume that the source signals, the elements of $\mathbf{s}(t)$ in equation

(2.1), are statistically independent at each time instant t. Note that this is not an unrealistic assumption and in practice this does not need to be completely true, as demonstrated in [27]. With this assumption, a method known as independent component analysis (ICA) can be used to estimate the elements of the mixing matrix and then, as a result of this, the source signals can be estimated. One clear advantage of using ICA is that it does not require any further knowledge of the different source signals or the positions of the sensors. It is also assumed when using this method that no more than one source signal is Gaussian, the sources have zero mean and that there are at least as many sensors as sources, i.e. in equation (2.1) $p \geq q$ [27]. With these assumptions, it is possible to recover the source signals subject to a couple of indeterminacies. Firstly, the order of the independent components (i.e. the estimated sources) cannot be determined and secondly, it is not possible to determine their energies (variances) and so the reconstructed signals might be multiplied by some scalar quantity. The second ambiguity can be removed by adding the constraint that each source must have unit variance, which can be taken into account by the ICA solution. The first indeterminacy cannot be removed however, but this is not a problem in many of the applications of the technique. A detailed description of ICA can be found in [3, 5, 27].

The majority of instantaneous BSS techniques implement a two-stage approach, where initially second-order statistics (SOS) are exploited to decorrelate and normalise the received signals, before the solution is completed using higher-order statistics (HOS) to obtain estimates of the source signals [28]. Note that SOS by themselves are not generally sufficient to enforce independence and therefore separate the sources. It is beyond the scope of this thesis to give a detailed review of the different algorithms for solving the instantaneous BSS problem. However, popular algorithms, which implement this two step approach, include Joint Approximation Diagonalisation of Eigenmatrices (JADE) [29], Second Order Blind Identification (SOBI) [30], Simultaneous Third Order Tensor Diagonalisation (STOTD) [31], BLInd Signal Separation (BLISS) [32], FastICA [33] and Comon's ICA methods discussed in [5]; a detailed review of the literature surrounding this topic can be found in [34].

The first stage of these two-stage algorithms can be carried out by calculating either the EVD of the covariance matrix of the received signals $\mathbf{x}(t)$ or alternatively by calculating the

SVD of a matrix containing all the time samples for the received data. Either decomposition yields a matrix capable of linearly transforming the observed signals to obtain a set of uncorrelated signals. This point will be discussed in the next section of this chapter, where three different decompositions of a scalar matrix and their applications to narrowband signal processing are discussed. This section has highlighted the necessity and value of two of these scalar matrix decompositions (the EVD and SVD) to narrowband signal processing. This has been discussed as an introduction to the more complicated convolutive case, which will be the initial application area of the equivalent polynomial matrix decompositions discussed in this thesis.

2.3 Decomposition Techniques for Scalar Matrices

The decompositions techniques for scalar matrices are very useful tools in linear algebra, as they can be used to simplify many numerical equations [6, 35–38]. The three decompositions examined in this thesis are the EVD, the SVD and the QRD and these are now discussed.

2.3.1 The QR Decomposition

The QRD aims to transform a matrix with complex scalar elements, into an upper triangular matrix by applying a series of unitary matrices [6, 36, 37]. The QRD of a scalar matrix $\mathbf{A} \in \mathbb{C}^{p \times q}$ is defined as

$$\mathbf{A} = \mathbf{Q}\mathbf{R} \tag{2.2}$$

where the matrix $\mathbf{R} \in \mathbb{C}^{p \times q}$ is an upper triangular matrix consisting of complex scalar entries and $\mathbf{Q} \in \mathbb{C}^{p \times p}$ is a unitary matrix, which means it satisfies $\mathbf{Q}^H\mathbf{Q} = \mathbf{Q}\mathbf{Q}^H = \mathbf{I}_p$. There are several different methods for calculating the decomposition, these include Givens rotations and Householder reflections. The details of how to formulate the decomposition using these methods are not discussed here, but an extensive description of each method can be found in [6]. Note that if $\mathbf{A}$ is non-singular, i.e. the matrix is square and of full rank, then the QRD of the matrix is unique, provided the diagonal elements are made to be both real

and positive [6]. Finally, as the transformation was performed by a unitary matrix, this decomposition is norm preserving, i.e. $\|\mathbf{A}\|_F^2 = \|\mathbf{R}\|_F^2$.

Applications of the QR Decomposition

The main advantage of the QRD is that it can often be used to enable set of linear equations to be easily solved. For example, consider the instantaneous mixing model demonstrated by equation (2.1) without the additive noise term for simplicity. Then provided the mixing matrix $\mathbf{C}$ is known, its QRD can be calculated such that $\mathbf{C} = \mathbf{QR}$, thus allowing the instantaneous mixing model to alternatively be expressed as

$$\mathbf{Q}^H\mathbf{x} = \mathbf{Rs} \tag{2.3}$$

The left hand-side of equation (2.3) can be calculated, transforming the system of equations $\mathbf{x} = \mathbf{Cs}$ into a triangular systems of equations, which are easier to solve for $\mathbf{s}$ given $\mathbf{C}$ and $\mathbf{x}$ using back substitution. This method is often used as the computations involved in calculating the QRD and then performing back substitution are less expensive than calculating the inverse $\mathbf{C}^{-1}$. However, this method of back substitution is not possible if the mixing matrix for the system is rank deficient, as this will lead to a number of diagonal elements of the upper triangular matrix $\mathbf{R}$ being equal to zero.

Another potential advantage of this decomposition is that it can also be used to calculate the eigenvalues of a square matrix. If the matrix $\mathbf{C}$ is square, then the diagonal elements of $\mathbf{R}$ are the eigenvalues of $\mathbf{C}$. Furthermore, the algorithm can be used to formulate an algorithm for determining the eigenvalues of a matrix, this algorithm is known as the QR algorithm and is often used when the matrix is not Hermitian.

2.3.2 Eigenvalue Decomposition

The eigenvalue decomposition (EVD) of the Hermitian matrix $\mathbf{R} \in \mathbb{C}^{p \times p}$ can be expressed as

$$\mathbf{R} = \mathbf{HDH}^H \tag{2.4}$$

where $\mathbf{H} \in \mathbb{C}^{p \times p}$ is a unitary matrix with columns equal to the orthonormal eigenvectors of $\mathbf{R}$, and $\mathbf{D} \in \mathbb{C}^{p \times p}$ is a diagonal matrix, whose diagonal elements are the corresponding eigenvalues of $\mathbf{R}$. The EVD can only be calculated for Hermitian matrices, but it does not matter if the entries of the matrix are real or complex. The unitary matrices perform similarity transformations, which means that the matrices $\mathbf{R}$ and $\mathbf{D}$ have the same eigenvalues. This decomposition is norm preserving and so $\|\mathbf{R}\|_F^2 = \|\mathbf{D}\|_F^2$. Note that to calculate the eigenvalues of a scalar matrix that is square, but not Hermitian, then the QRD can be used instead.

The EVD can be calculated using Jacobi's algorithm, which operates by applying a series of Jacobi rotations [6]. This method is very popular as it is inherently parallel. However, for large matrices where $p > 10$, this algorithm is much slower than a QR method. The details of how the algorithm operates will not be discussed here, for a detailed explanation refer to [6, 38]. Alternatively, the matrix $\mathbf{R}$ can be first reduced to a tridiagonal matrix using either Givens rotations or Householder reflections. Both techniques are stable, but the Householder method is a factor of two more efficient [38]. Note that if the matrix is not of full rank then at least one eigenvalue is equal to zero.

Applications of the EVD

The EVD for scalar matrices is extensively used in DSP. For example, the unitary matrix obtained by calculating the EVD is the Karhunen-Loeve transform used for optimal data compaction and is formulated by calculating the EVD of the covariance matrix of the set of data signals to be coded [7, 16]. The most significant application for this thesis, is that the EVD can be used to decorrelate a set of instantaneously mixed signals. This will now be discussed.

Decorrelation of Instantaneously Mixed Signals Using the EVD

The covariance matrix of the observed signals $\mathbf{x}(t)$ from equation (2.1), which are assumed to be zero-mean jointly wide sense stationary (WSS), can be calculated as

$$\begin{aligned}
\mathbf{R}_{xx} &= E\left[\mathbf{x}(t)\mathbf{x}(t)^H\right] && (2.5)\\
&= E\left[\mathbf{C}\mathbf{s}(t)(\mathbf{C}\mathbf{s}(t))^H\right] + \sigma^2\mathbf{I}_p && (2.6)\\
&= \mathbf{C}E\left[\mathbf{s}(t)\mathbf{s}(t)^H\right]\mathbf{C}^H + \sigma^2\mathbf{I}_p && (2.7)\\
&= \mathbf{C}\mathbf{R}_{ss}\mathbf{C}^H + \sigma^2\mathbf{I}_p && (2.8)
\end{aligned}$$

where $\mathbf{R}_{ss} \in \mathbb{C}^{q\times q}$ denotes the spatial covariance matrix of the source signals [39]. As the source signals are assumed to be statistically independent, their cross-correlation terms will equal zero and, as a result, the covariance matrix $\mathbf{R}_{ss}$ will be diagonal. Furthermore, if they have unit power then $\mathbf{R}_{ss} = \mathbf{I}_q$. However, the covariance matrix of the observed signals $\mathbf{R}_{xx} \in \mathbb{C}^{p\times p}$ will generally not be diagonal, as the observed signals constitute a linear combination of the source signals and are therefore correlated with one another.

The process of whitening the observed signals transforms them so that they are uncorrelated and therefore have a diagonal covariance matrix. This can be achieved by calculating the EVD of the covariance matrix of the mean-removed observed signals. To demonstrate this point, define the data matrix containing the observed samples $\mathbf{x}(t)$ for $t = 0, 1, \ldots, T-1$, using the notation

$$\mathbf{X} = [\mathbf{x}(0), \mathbf{x}(1), \ldots, \mathbf{x}(T-1)]. \tag{2.9}$$

Using this matrix a sample estimate of the true covariance matrix $\mathbf{R}_{xx}$ of the observed signals can be calculated as

$$\widehat{\mathbf{R}}_{xx} = \frac{\mathbf{X}\mathbf{X}^H}{T}. \tag{2.10}$$

This matrix is Hermitian and so its EVD can be calculated such that

$$\mathbf{U}\widehat{\mathbf{R}}_{xx}\mathbf{U}^H = \mathbf{D}, \tag{2.11}$$

where $\mathbf{U} \in \mathbb{C}^{p \times p}$ is a unitary matrix and $\mathbf{D} \in \mathbb{C}^{p \times p}$ is a diagonal matrix whose diagonal elements satisfy $d_{11} \geq \ldots \geq d_{qq}$. Note that these elements correspond to the estimated powers of the source signals.

Subsequently, the transformed data matrix

$$\mathbf{X}' = \mathbf{U}\mathbf{X} \tag{2.12}$$

will now contain the series of signals $\mathbf{x}'(0), \ldots, \mathbf{x}'(T-1)$, with an estimated covariance matrix equal to the diagonal matrix $\mathbf{D}$ and which are not correlated with one another. The signals are said to have been instantaneously decorrelated and the transformed signals placed in order of decreasing power. The application of the unitary matrix $\mathbf{U}$ in equation (2.12) will modify the phase and amplitude of the observed signals stored in the data matrix $\mathbf{X}$ and the signal and noise subspaces will have been separated, which is very useful for adaptive beamforming and high resolution direction finding [7, 25].

Note that second order statistics can only be used to decorrelate the signals and are not generally sufficient to enforce independence of the signals. Instead HOS are required to complete the solution and so an ICA algorithm must now be applied to the whitened data to reconstruct the independent source signals. However, if the received signals each have very different power levels, then the majority of the separating of the sources can be achieved by using either the EVD or SVD [7]. Note that this preprocessing step has reduced the problem from estimating p^2 parameters to one of $p(p-1)/2$ degrees of freedom [27] and has therefore significantly simplified the problem.

2.3.3 The Singular Value Decomposition

The singular value decomposition (SVD), unlike the EVD, can be applied to both square and rectangular matrices to transform a matrix of complex scalar entries into a diagonal matrix and the matrix to be factorised need not be Hermitian [6, 40]. The SVD of the matrix $\mathbf{A} \in \mathbb{C}^{p \times q}$, where the elements of the matrix can be either real or complex scalars, is defined

as

$$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{H} \tag{2.13}$$

where the matrices $\mathbf{U} \in \mathbb{C}^{p\times p}$ and $\mathbf{V} \in \mathbb{C}^{q\times q}$ are both unitary and $\boldsymbol{\Sigma} = diag(\sigma_1, \ldots, \sigma_N) \in \mathbb{R}^{p\times q}$. The diagonal entries of $\boldsymbol{\Sigma}$ are referred to as the singular values of the matrix $\mathbf{A}$ and will satisfy $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_N \geq 0$, where $N = \min\{p, q\}$. The columns of the unitary scalar matrix $\mathbf{U}$ contain the left singular vectors of $\mathbf{A}$ and form an orthonormal basis. Similarly, the columns of $\mathbf{V}$ define the right singular vectors. As with the previous two decomposition, this decomposition is norm preserving and so $\|\mathbf{A}\|_F^2 = \|\boldsymbol{\Sigma}\|_F^2$.

Relationship Between the EVD and SVD

Suppose the SVD of the matrix $\mathbf{A} \in \mathbb{C}^{p\times q}$ has been calculated according to equation (2.13). Then using this decomposition the matrices $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$ can be calculated as follows

$$\mathbf{A}\mathbf{A}^H = \mathbf{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H\mathbf{U}^H \tag{2.14}$$

and

$$\mathbf{A}^H\mathbf{A} = \mathbf{V}\boldsymbol{\Sigma}^H\boldsymbol{\Sigma}\mathbf{V}^H, \tag{2.15}$$

which are the EVDs of the matrices $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$ respectively. The matrices $\boldsymbol{\Sigma}\boldsymbol{\Sigma}^H = diag\{\sigma_1^2, \ldots, \sigma_p^2\} \in \mathbb{C}^{p\times p}$ and $\boldsymbol{\Sigma}^H\boldsymbol{\Sigma} = diag\{\sigma_1^2, \ldots, \sigma_q^2\} \in \mathbb{C}^{q\times q}$. The unitary matrices $\mathbf{U}$ and $\mathbf{V}$ required for the singular value decomposition of the matrix $\mathbf{A}$ could therefore have been obtained by calculating the EVD of the matrices $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$.

Applications of the SVD

The SVD can be used to diagonalise a scalar matrix as demonstrated by equation (2.13) and can accordingly be used to simplify a set of linear equations in a similar way to the QRD. Furthermore, the decomposition can also be used to determine the rank, range and null space of a matrix and is also used when calculating the pseudo-inverse of a matrix [6]. This decomposition therefore has a vast range of applications and has become a popular numerical

tool in many areas of research, such as statistical data analysis, control system analysis, scientific computing, system identification and also signal processing [41]. For example, the SVD is the method of choice for solving the linear least squares problem, where it can be applied directly to the data matrix to obtain a robust solution [25] and can be applied to problems of image restoration and compression [42]. The SVD can also be used to decorrelate a set of instantaneously mixed signals, where it can be used as an alternative technique to the EVD. For this application, the SVD is preferred to the EVD as it is less computationally expensive. This will now be discussed further.

Decorrelation of Instantaneously Mixed Signals Using the SVD

It has previously been demonstrated that the unitary matrix obtained from calculating the EVD of the covariance matrix of a set of instantaneously mixed signals can be used to decorrelate the set of received signals as demonstrated in equation (2.12). Note that the source signals are assumed to be uncorrelated and stationary. Alternatively, the SVD of the data matrix $\mathbf{X} \in \mathbb{C}^{p \times q}$ from equation (2.9) could be calculated directly such that

$$\mathbf{U}\mathbf{X}\mathbf{V}^{\boldsymbol{H}} = \boldsymbol{\Sigma} \tag{2.16}$$

where $\mathbf{U} \in \mathbb{C}^{p \times p}$ and $\mathbf{V} \in \mathbb{C}^{q \times q}$ are unitary matrices and $\boldsymbol{\Sigma} \in \mathbb{C}^{p \times q}$ is a diagonal matrix whose diagonal coefficients satisfy $\sigma_{11} \geq \ldots \geq \sigma_{qq} \geq 0$. The relationship between the EVD of the covariance matrix and SVD of the data matrix is easily seen as follows

$$(\mathbf{U}\mathbf{X}\mathbf{V}^{H})(\mathbf{U}\mathbf{X}\mathbf{V}^{H})^{H} = \Sigma^{2}. \tag{2.17}$$

Hence

$$\mathbf{U}\mathbf{X}\mathbf{X}^{H}\mathbf{U}^{H} = \Sigma^{2} \tag{2.18}$$

and so

$$\mathbf{U}\hat{\mathbf{R}}_{xx}\mathbf{U}^{H} = \frac{\Sigma^{2}}{T} \equiv \mathbf{D}. \tag{2.19}$$

This has demonstrated that the matrix $\mathbf{U}$ obtained from calculating the SVD of the data

matrix $\mathbf{X}$ is also sufficient to impose decorrelation upon the set of received signals. Note that the SVD computation is preferable in terms of arithmetic precision [6]. Once again, the total energy of the signals is preserved under the transformation, i.e. $\sum_{i=0}^{T-1} \|\mathbf{x}(i)\|_2^2 = \text{trace}(\mathbf{X}\mathbf{X}^H) = \text{trace}(\mathbf{\Sigma}\mathbf{\Sigma}^H) = \sum_{i=1}^{q} \sigma_{ii}^2 \equiv T \sum_{i=1}^{q} d_{ii}$.

2.4 Convolutive Mixtures

The instantaneous mixing model shown in equation (2.1) is not suitable for many realistic situations where the propagation of the signals from the sources to the sensors can take multiple paths with different time delays. In this situation polynomial matrices are required to describe the mixing and so each element of the mixing matrix will be a finite impulse response (FIR) filter with an associated set of coefficients. Before discussing the convolutive mixing model, polynomial matrices and some properties associated with polynomial matrices are discussed. In this thesis, it is assumed that the term polynomial includes Laurent polynomials, which allow for negative powers of the interdeterminate variable of the polynomial.

2.4.1 Polynomial Matrices

A polynomial matrix is simply a matrix with polynomial elements. However, it can alternatively be thought of as a polynomial with matrix coefficients and so a $p \times q$ polynomial matrix $\underline{\mathbf{A}}(z)$, where the indeterminate variable of the polynomial is z^{-1} (used in the context of this thesis to represent a unit delay), can be expressed as

$$\underline{\mathbf{A}}(z) = \sum_{\tau=t_1}^{t_2} \mathbf{A}(\tau) z^{-\tau} = \begin{bmatrix} \underline{a}_{11}(z) & \underline{a}_{12}(z) & \cdots & \underline{a}_{1q}(z) \\ \underline{a}_{21}(z) & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \underline{a}_{p1}(z) & \cdots & \cdots & \underline{a}_{pq}(z) \end{bmatrix}, \tag{2.20}$$

where $\tau \in \mathbb{Z}$ and $t_1 \leq t_2$. The **order** of this polynomial matrix is calculated as $(t_2 - t_1)$, where the values of the parameters t_1 and t_2 are not necessarily positive. The matrices

$\mathbf{A}(t_1), \ldots, \mathbf{A}(t_2)$, which will generally have complex scalar entries, are referred to as the set of coefficient matrices for the polynomial matrix $\underline{\mathbf{A}}(z)$. In particular, the coefficient matrix $\mathbf{A}(t)$ will be referred to as the coefficient matrix of order t. Note that the coefficient matrix of order zero, i.e. $\mathbf{A}(0)$, is particularly important for the discussion of the polynomial matrix decomposition algorithms within this thesis. Using the notation outlined in Section 1.4, the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$ as all coefficients of the polynomial elements of the matrix are complex. Alternatively, $\underline{\mathbf{A}}(z) \in \mathbb{C}^{p \times q \times (t_2 - t_1)}$ if the order of the matrix is also known.

2.4.2 Properties of a Polynomial Matrix

The **paraconjugate** of the polynomial matrix $\underline{\mathbf{A}}(z)$ is defined to be

$$\widetilde{\underline{\mathbf{A}}}(z) = \underline{\mathbf{A}}_*^T(1/z) \tag{2.21}$$

where * denotes the complex conjugation of the coefficients of each polynomial element and ^{T} denotes matrix transposition. The tilde notation as used in the above expression will be used throughout this thesis to denote paraconjugation of a polynomial matrix. A polynomial matrix $\underline{\mathbf{A}}(z)$ is said to be **paraunitary** if the following is true

$$\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z) = \widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) = \mathbf{I}. \tag{2.22}$$

Some definitions, for example in [16], define a paraunitary matrix if it satisfies $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) = c^2\mathbf{I}$, however, this will not be used in this thesis and so instead it is assumed that a matrix is paraunitary if it satisfies equation (2.22). Note that a paraunitary polynomial matrix represents a multi-channel all-pass filter and, accordingly, it preserves the total signal power at every frequency [7, 16]. Furthermore, note that the product of paraunitary matrices will also be paraunitary and will satisfy

$$\widetilde{\underline{\mathbf{A}}(z)\underline{\mathbf{B}}}(z) = \widetilde{\underline{\mathbf{B}}}(z)\widetilde{\underline{\mathbf{A}}}(z). \tag{2.23}$$

A polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times p}$ is **para-Hermitian** if it is equal to its paraconjugate,

i.e. if

$$\underline{\widetilde{\mathbf{A}}}(z) = \underline{\mathbf{A}}(z) \tag{2.24}$$

and so the individual coefficients associated with the polynomial elements satisfy $a_{jk}(t) = a^*_{kj}(-t)$ $\forall t \in \mathbb{Z}$ and for $j, k = 1 \ldots, p$.

The **degree** of a polynomial matrix is the minimum number of delays units required to implement the polynomial matrix. For example, the polynomial matrix

$$\underline{\mathbf{A}}(z) = \begin{bmatrix} z^{-1} & 0 \\ 0 & z^{-1} \end{bmatrix} \tag{2.25}$$

has degree two [16]. Note that this is not the same as the order of the polynomial matrix, which for this example is equal to one.

Finally, the **Frobenius norm** of the polynomial matrix $\underline{\mathbf{A}}(z)$ is defined to be

$$\|\underline{\mathbf{A}}(z)\|_F = \sqrt{\sum_{\tau=t_1}^{t_2} \sum_{i=1}^{p} \sum_{j=1}^{q} |a_{ij}(\tau)|^2} \tag{2.26}$$

This can also be expressed as

$$\|\underline{\mathbf{A}}(z)\|_F = \sqrt{trace\left(\left[\underline{\mathbf{A}}(z) \underline{\widetilde{\mathbf{A}}}(z) \right] \Big|_0 \right)} \tag{2.27}$$

where $[\cdot]|_0$ defines the coefficient matrix of z^0 in the polynomial matrix.

2.4.3 The Mixing Model

For the simpler instantaneously mixing model demonstrated by equation (2.1), each of the received signals consists of a sum of differently weighted source signals, all sampled at the same instant in time. However, for convolutively mixed signals the model required to express the mixing is more complex as the received signals now consist of weighted and delayed versions of the source signals. This can be due to the sources arriving at the sensors over multiple paths and with different time delays, where the delays observed can either be due to finite propagation speed in the medium through which the sources are traveling or possibly

reverberations from obstacles in its path, i.e. multipath propagation.

It is assumed that a set of source signals $\mathbf{s}(t) \in \mathbb{C}^{q\times 1}$ where $t \in \{0,\ldots,T-1\}$ are emitted from q independent sources through a convolutive channel, to be received at an array of p sensors, where it is assumed that there are at least as many sensors as sources, i.e. $p \geq q$. The relationship between the source signals and convolved received signals, $\mathbf{x}(t) \in \mathbb{C}^{p\times 1}$, where $t \in \{0,\ldots,T-1\}$, can be expressed by the convolutive mixing model

$$\mathbf{x}(t) = \sum_{k=0}^{N} \mathbf{C}(k)\mathbf{s}(t-k) + \mathbf{n}(t) \tag{2.28}$$

where $\mathbf{n}(t) \in \mathbb{C}^{p\times 1}$ denotes an additive Gaussian noise process with variance $\sigma^2\mathbf{I}$ and $\mathbf{C}(k) \in \mathbb{C}^{p\times q}$ for $k \in \{0,\ldots,N\}$ denote the coefficient matrices of the polynomial mixing matrix. The polynomial mixing (or channel) matrix for the model can also be denoted as

$$\underline{\mathbf{C}}(z) = \sum_{k=0}^{N} \mathbf{C}(k)z^{-k}, \tag{2.29}$$

where the order of this matrix will be N. The mixing model of equation (2.28) can therefore also be written in the form

$$\underline{\mathbf{x}}(z) = \underline{\mathbf{C}}(z)\underline{\mathbf{s}}(z) + \underline{\mathbf{n}}(z), \tag{2.30}$$

where $\underline{\mathbf{x}}(z)$, $\underline{\mathbf{s}}(z)$ and $\underline{\mathbf{n}}(z)$ each represent algebraic power series of the form $\underline{\mathbf{x}}(z) = \sum_{\tau=-\infty}^{\infty} \mathbf{x}(t)z^{-t}$ of the received signals, the source signals and the noise respectively. This is the more realistic of the two mixing scenarios and arises in many real-world situations. For example, convolutively mixed signals will be observed in a teleconferencing environment, where audio signals are produced in a reverberant room. They are also observed in a digital communication environment, where there are multiple transmit antennas operating at the same radio frequency and the transmitted signals are received at multiple receive antennas.

Note that if the number of sources in equation (2.28) exceeds the number of sensors and so $p < q$, then the problem is said to under-determined and linear methods of source separation will generally not be able to recover the sources, even if there is perfect knowledge of the mixing matrix. For this reason, this case is not considered in this thesis.

2.4.4 Convolutive Source Separation

Deconvolution is the problem of ascertaining the source signals with full knowledge of the received signals and the linear time-invariant (LTI) system, i.e. the polynomial channel matrix. Blind deconvolution, or blind equalisation as it is also known, is the problem of finding the source signals without any specific prior knowledge of the source signals or the mixing matrix for the system.

The ability to deconvolve the received signals of equation (2.28) to obtain estimates of the source signals has many applications. Over the last decade, BSS of convolutive mixtures has been studied extensively, with many of the existing methods for solving the problem simply derived as extensions of existing algorithms designed for the instantaneous situation. The collection of methods for unmixing the convolved signals can be divided into two groups; those that operate in the frequency domain and those that are used in the time domain.

Time Domain Approach

Most time domain convolutive BSS algorithms do not operate using a two-stage method using SOS and then HOS, as implemented with most instantaneous algorithms. Instead, popular methods are gradient descent [43] and neural networks [44]. Methods based on cost functions are frequently used, such as, for example, the stochastic gradient optimisation technique, which employs the use of SOS and HOS simultaneously [45]. Other existing methods for BSS of convolved signals include Bussgang, least squares lattice prediction and linear blind deconvolution filters [3, 25].

Alternatively, other algorithms operating in the time domain, use SOS to strongly decorrelate the signals and then exploit HOS to identify and apply the hidden paraunitary matrix required to complete the source separation [8, 22]. Decorrelation, using either the SVD or EVD, has been shown to be a useful preprocessing step for instantaneous blind source separation. However, with convolutively mixed signals, they are not only correlated with each other at the same time instant, but possibly over a range of time delays as well. Instead a polynomial matrix is required to transform the received signals and impose strong decorrelation. The EVD or SVD of a polynomial space-time covariance matrix is required to obtain

this transformation matrix, which is capable of enforcing strong decorrelation upon the signals. This decomposition cannot be formulated using the conventional scalar matrix EVD or SVD discussed in Section 2.3, instead a polynomial matrix decomposition method is required. Subsequently, to complete the solution, some HOS cost function, typically based on fourth order statistics, is optimised. Methods for completing the second step are beyond the scope of this thesis, but a detailed review can be found in [45, 46].

Frequency Domain Approach

One approach to frequency domain processing is to use the discrete Fourier transform (DFT) to split the convolutively mixed data into narrower frequency bands. For each frequency $\omega = 2\pi f$, the convolutive mixing process demonstrated in equation (2.28) may be expressed in the form

$$\mathbf{X}(\omega) = \mathbf{C}(\omega)\mathbf{S}(\omega) + \mathbf{N}(\omega) \tag{2.31}$$

where $\mathbf{C}(\omega) \in \mathbb{C}^{p\times q}$ and has complex scalar entries, $\mathbf{X}(\omega) \in \mathbb{C}^{p\times 1}$, $\mathbf{N}(\omega) \in \mathbb{C}^{p\times 1}$ and $\mathbf{S}(\omega) \in \mathbb{C}^{q\times 1}$. It can be demonstrated that each Fourier component of the received data is a complex scalar mixture of the corresponding Fourier components of the source signals [3, 8, 46, 47]. Consequently, the broadband problem has been reduced to a series of narrowband problems, where an instantaneous BSS method can then be employed.

This method is particulary useful if the time domain filters are long, i.e. the order of the polynomial channel matrix in equation (2.30) is large, as is often observed in acoustic problems [45]. However, this method does have its problems. Firstly, the problem of permutation that was observed with the instantaneous methods, will exist within the individual solutions for each frequency band and these permutations, typically, will not be the same in each band. Consequently, when the signals are converted back to the time domain, contributions from different sources can then be remixed into a single channel. There are methods for solving this problem, however they require further assumptions to be placed on either the signals or the mixing environment. Similarly, the scale problem associated with the instantaneous BSS techniques will again be present within each of the frequency band solutions, although this problem is more easily solved by normalisation. There are advantages and disadvantages of

using both time and frequency domain approaches, which are detailed extensively in [45].

A set of convolutively mixed signals could be strongly decorrelated by reducing the problem to narrowband form as demonstrated by equation (2.31) and then the SVD of each narrowband problem could be calculated. However, this technique will ignore any correlations that exist between different frequency bands. Furthermore, the SVD in each frequency band will order the output channels in terms of decreasing power, irrespective of the ordering of neighboring channels [7] and this can lead to incoherence between the different narrowband problems. However, the method has had success in the context of space-time adaptive processing for phased array radar [48].

Application to MIMO Communications

In communication applications, the system described by equation (2.28) is referred to as a Multi-Input Multi-Output (MIMO) system. In this situation, data is transmitted from multiple transmit antennas, which for this example is equal to q. This data then passes through the propagation channel, which in a realistic scenario will take multiple paths and different time delays, before being received by multiple receive antennas, whose number is represented by p. These multipath systems arise due to scattering, reflection, refraction or diffraction of radiated energy off any objects that lie in the environment [2, 49]. The use of this system offers the advantage of improved communication performance, by making use of the multiple transmitters and receivers to provide array, diversity and/or multiplexing gain(s) [2].

Equalisation is the process of recovering a signal that has been corrupted by a multipath environment from a single received signal. Multichannel equalisation is the same process, however, there are now multiple mixed signals to be equalised. The problem is termed blind, if the user only has access to the received (convolved) signals. The term blind is neglected, if the user also has access to the channel matrix for the system.

In this scenario, the polynomial channel matrix for the system is typically known as it has been previously been estimated. The problem requires that all channels and cross-channels in a problem are equalised to obtain an estimate of the transmitted signals. This application is

discussed extensively in Chapter 8, where the polynomial matrix decomposition techniques, which are introduced in this thesis, have been applied to simplify the problem prior to the equalisation step. By using these decompositions there is no longer any need for the cross-channels to be equalised, leaving only a set of single channel equalisation problems to be solved.

2.5 Existing Polynomial Matrix Decompositions

Methods do exist for calculating decompositions or factorisations of polynomial matrices. For example, these methods include the Smith-McMillan decomposition [16, 19] and a method developed by Vaidyanthan for factorising a paraunitary polynomial matrix into a series of elementary rotations and delays [16]. However, little research has been done prior to the Sequential Best Rotation algorithm (SBR2) [7], surrounding the EVD generalisation for polynomial matrices in the time domain. In [20] Lambert reports extensive work on the separation of broadband signals and claims that he has developed an EVD routine suitable for polynomial matrices. He represents convolved signals in terms of DFT filter matrices and polynomial matrices. However, Lambert's PEVD method entails the inversion of FIR filters in the frequency domain and is therefore very different from the SBR2 algorithm. To the best of our knowledge, there are currently no other existing methods for directly calculating the QRD of a polynomial matrix. The existing techniques for achieving some form of polynomial matrix decomposition are now discussed.

2.5.1 FIR Lossless System Decomposition

In [16] Vaidyanathan introduces a method for factorising any finite degree paraunitary polynomial matrix, such as the matrix describing a lossless FIR system, into a series of paraunitary matrices comprised of simple rotation and delay matrices. The process operates as a series of steps, where at each step a Givens rotation [6] and an elementary delay matrix can be factored out of the polynomial matrix representing the system $\underline{\mathbf{H}}_N(z)$ of degree N, resulting in a paraunitary matrix where the degree of the system has been reduced by unity. The

factorisation relies on the fact that the determinant of a paraunitary matrix is equal to a single delay, i.e. is of the form az^{-k} where $a \neq 0$ and $k \geq 0$ is an integer. At each stage of the process, the system is reduced by factoring out a Givens rotation and an elementary delay matrix, which are both paraunitary.

As a simple example, suppose the polynomial paraunitary matrix $\underline{\mathbf{H}}_N(z)$ describes a 2×2 real coefficient FIR lossless system and has degree N. The first step of Vaidyanathan's factorisation routine is to formulate a Givens rotation matrix $\mathbf{Q}_N$ and a delay matrix $\underline{\mathbf{\Lambda}}(z)$, which is a matrix of the form

$$\underline{\mathbf{\Lambda}}(z) = \begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix} \tag{2.32}$$

and when applied to another polynomial matrix it will impose a unit delay upon one channel, such that

$$\underline{\mathbf{H}}_N(z) = \underbrace{\begin{bmatrix} cos(\theta_N) & sin(\theta_N) \\ -sin(\theta_N) & cos(\theta_N) \end{bmatrix}}_{=\mathbf{Q}_N} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & z^{-1} \end{bmatrix}}_{=\underline{\mathbf{\Lambda}}(z)} \underline{\mathbf{H}}_{N-1}(z) \tag{2.33}$$

where $\underline{\mathbf{H}}_{N-1}(z)$ is also an FIR lossless matrix, but the degree of the determinant of this transformed matrix will have been reduced by unity and so the degree of the system has been reduced. Following this step it is said that a degree one block has been extracted. This process is repeated until a degree zero block is found. Therefore, the overall factorisation of $\underline{\mathbf{H}}_N(z)$ can be expressed as

$$\underline{\mathbf{H}}_N(z) = \mathbf{Q}_N \underline{\mathbf{\Lambda}}(z) \ldots \underline{\mathbf{\Lambda}}(z) \mathbf{Q}_1 \underline{\mathbf{\Lambda}}(z) \mathbf{Q}_0 \Psi \tag{2.34}$$

where Ψ is a diagonal matrix with unimodular elements and must be included to allow for the ambiguity in the problem due to a possible permutation and multiplication by ± 1. The overall process to decompose the 2×2 real coefficient FIR lossless system is illustrated in Figure 2.1, where α is again required to account for this ambiguity. Note that the overall transformation will be paraunitary by construction, as the individual matrices $\mathbf{Q}_m$ and $\underline{\mathbf{\Lambda}}(z)$ are parauntiary, for $m = 1, \ldots, N$, and so their product will also be paraunitary. After each stage, the determinant of the reduced system will be reduced by unity.

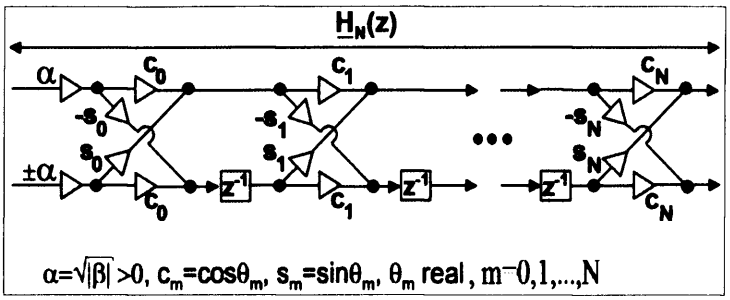

Figure 2.1: The factorisation of a 2×2 degree N paraunitary polynomial matrix into a series of delay and rotation blocks, according to Vaidyanthan's FIR lossless system decomposition method [16].

Vaidyanthan then generalises this statement and explains that any paraunitary polynomial matrix of any dimension, for example $\underline{\mathbf{H}}_N(z) \in \mathbb{R}^{p \times p}$, can be decomposed as shown by equation (2.34) provided it is of fixed degree. Now $\mathbf{Q}_m \in \mathbb{R}^{p \times p}$ for $m = 0, \ldots, N$ and $\underline{\mathbf{\Lambda}}(z) \in \underline{\mathbb{R}}^{p \times p}$ is the identity matrix with the exception of the p^{th} diagonal element, which will be z^{-1}. The process can also be easily extended to deal with a paraunitary matrix, whose polynomial coefficients are complex, by using complex Givens rotations.

The main problem with this decomposition technique is that it only ever implements a unit delay in one step, which is not practical when the paraunitary matrix to be calculated is quite simple, yet of a high order. For example, if the degree of the paraunitary polynomial matrix to be factorised is 5, but it has only two non-zero coefficient matrices ($\mathbf{H}_5(0)$ and $\mathbf{H}_5(5)$) then this decomposition of the paraunitary matrix would require five stages. This is discussed further in [8]. Note that this would not be a problem with the SBR2 algorithm discussed in Chapter 3.

In [50] Regalia and Huang also derive a method for calculating a two-channel lossless FIR filter for optimal data compression using the fixed degree parameterisation scheme proposed by Vaidyanathan, which is discussed in Section 2.5.1. This technique can be used to determine the optimal paraunitary matrix required for computing the EVD of a 2×2 paraunitary polynomial matrix. The applications of this decomposition lie in subband coding and wavelet signal analysis and are briefly discussed towards the end of the thesis in the Section 8.5.4.

2.5.2 Smith Decomposition

Given a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$, it is possible to obtain simpler forms of this matrix, such as diagonal, upper or lower triangular polynomial matrix, by performing elementary operations upon this polynomial matrix [16, 51]. The elementary operations are similar to those applicable to scalar matrices and come in both row and column forms [6]. For example, the elementary row operations are defined as

1. Interchange two rows of the polynomial matrix
2. Scale one row of the polynomial matrix by a nonzero constant c
3. Add a polynomial multiple of one row to another.

The column operations are defined similarly and the matrices capable of implementing the three row operations are referred to as elementary matrices. Note that all three types of elementary matrices are known as unimodular matrices. A unimodular polynomial matrix is a square polynomial matrix with a constant nonzero determinant. It is easily deduced from this property that a polynomial matrix is only unimodular if and only if its inverse is a polynomial matrix, which will also be unimodular [16]. Note, however, that the elementary matrices are not generally paraunitary so the Smith decomposition is quite distinct from a polynomial matrix EVD or SVD.

By repeated application of a finite number of elementary operations (pre-multiplication to make row operations and post-multiplication to perform column operations) the polynomial matrix $\underline{\mathbf{A}}(z)$ can be reduced to a diagonal polynomial matrix $\underline{\Gamma}(z) \in \underline{\mathbb{C}}^{p\times q}$ as follows

$$\underline{\mathbf{W}}(z)\underline{\mathbf{A}}(z)\underline{\mathbf{V}}(z) = \underline{\Gamma}(z) \tag{2.35}$$

where $\underline{\mathbf{W}}(z) \in \underline{\mathbb{C}}^{p\times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q\times q}$ both denote unimodular polynomial matrices. This is referred to as the Smith form decomposition. Note that the overall transformation matrix is unimodular as it consists of a series of unimodular matrices. In fact, one result presented in [16], is that any unimodular matrix can be formulated as the product of elementary matrices. Finally, the matrix $\underline{\mathbf{A}}(z)$ does not need to be square, although both the unimodular matrices

required to post- and pre-multiply the matrix to transform it to a diagonal matrix will be square.

The matrix $\underline{\Gamma}(z)$ is called the Smith form of $\underline{\mathbf{A}}(z)$ and is unique. Furthermore the diagonal elements of this matrix are calculated as follows

$$\underline{\Gamma}_{ii}(z) = \frac{\underline{\gamma}_{i+1}(z)}{\underline{\gamma}_i(z)}, \tag{2.36}$$

where $\underline{\gamma}_i(z)$ is the greatest common divisor or all the $i \times i$ minors of the polynomial matrix $\underline{\mathbf{A}}(z)$ and $\underline{\gamma}_0(z) = 1$.

The decomposition can be used to formulate an irreducible Matrix Fraction Description (MFD) of a polynomial transfer function, which can then be used for calculating the poles and zeros of this polynomial matrix [19]. This is of use in communications, where the Smith form of a transfer function of a MIMO system can be calculated and is used to understand the characteristics of the multivariable system.

2.5.3 Smith-McMillan Form

Suppose $\underline{\mathbf{H}}(z) \in \underline{\mathbb{C}}^{p \times q}$ is the transfer function of a MIMO system with polynomial elements $\underline{h}_{jk}(z) = \underline{a}_{jk}(z)/\underline{d}(z)$ where $j = 1, \ldots, p$, $k = 1, \ldots, q$ and $\underline{d}(z)$ is the least common multiple of the individual rational transfer functions. Then if the polynomial matrix $\underline{\mathbf{A}}(z)$ has the Smith decomposition given by equation (2.35), it follows that

$$\underline{\mathbf{W}}(z)\underline{\mathbf{H}}(z)\underline{\mathbf{V}}(z) = \underline{\Lambda}(z) \tag{2.37}$$

and this is called the Smith-McMillan decomposition of the polynomial matrix $\underline{\mathbf{H}}(z)$.

2.5.4 Lambert's FIR Matrix Eigenroutine

In [20] Lambert introduces a numerical routine for calculating the eigenvalues and eigenvectors of an FIR polynomial matrix by generalising conventional linear algebra and control techniques from the complex number field to the field of rational functions. The routine

is introduced to operate on a (2×2) polynomial matrix, but could easily be extended to operate on larger matrices, by applying the routine to a series of (2×2) sub-matrices of the matrix to obtain a diagonal matrix. The routine operates by transforming the elements of a polynomial matrix to the frequency domain using the DFT. Householder reflections are then used to obtain a upper triangular form of the matrix, where the eigenvalues of the matrix will be the diagonal elements of this matrix.

Lambert has demonstrated in his thesis [20] that his FIR matrix eigenroutine can be used as a whitening or preprocessing step to some least squares adaptation methods where multipath propagation of the signals has been observed. This whitening step will typically result in improved convergence of these algorithms. He states that it is always best to prewhiten the data, however, this is not always possible as it will require off-line computation [52, 53]. Note that since Lambert's PEVD routine entails the inversion of FIR filters in the frequency domain, it is very different from the SBR2 algorithm.

2.6 Conclusions

This chapter has given a brief overview of the background to polynomial matrix decompositions, in particular explaining how polynomial matrices arise in signal processing when a set of signals are received at an array of sensors over multiple paths and with different time delays. This is referred to as convolutive mixing and the mixing matrix required to express this takes the form of a polynomial matrix, where each element is a finite impulse response (FIR) filter. Existing techniques and numerical procedures for obtaining polynomial matrix decompositions have been discussed. Note that there are no existing techniques for achieving the polynomial matrix decompositions proposed in this thesis. The next chapter discusses the SBR2 algorithm, whose methodology forms the basis of the research presented in this thesis. It also demonstrates how this algorithm can be used to achieve strong decorrelation of a set of convolutively mixed signals, demonstrating that the applications of polynomial matrix decompositions are often simple extensions of scalar matrix decompositions from narrowband to broadband signal processing.

This chapter has also given a detailed overview of some scalar matrix decompositions (the EVD, SVD and QRD) and discussed their potential applications. This thesis is concerned with extending these matrix decompositions to polynomial matrices. Although the research has been motivated by the potential applications of polynomial matrix decompositions to signal processing, the overall objective is to demonstrate that other polynomial matrix decompositions can be formulated using the techniques proposed in developing the SBR2 algorithm.

Chapter 3

SBR2: A Polynomial Eigenvalue Decomposition Technique

3.1 Introduction

This chapter discusses an algorithm known as SBR2 for computing the Eigenvalue Decomposition (EVD) of a para-Hermitian polynomial matrix, [7]. The SBR2 algorithm was initially developed as the preliminary (second order) stage of a multistage blind signal separation (BSS) algorithm suitable for convolutive mixtures. The subsequent (higher order) stage takes the form of a sequential best rotation (SBR) algorithm, which exploits only fourth order statistics and is therefore referred to as SBR4 [8, 54]. Just as many instantaneous BSS algorithms use the EVD or SVD as a second order preprocessing step, the SBR2 algorithm can be applied to broadband mixtures, where polynomial matrices are now observed, before applying a convolutive BSS algorithm requiring HOS. Note that the polynomial matrix EVD (PEVD) algorithm can also be used in its own right as a time-domain approach for strongly decorrelating a set of convolutively mixed signals and for identifying the different signal and noise subspaces, [21].

The PEVD algorithm is referred to as SBR2 since it adopts an SBR strategy, but only involves the manipulation of second order statistics. It operates by applying a series of paraunitary similarity[1] transformation matrices to a para-Hermitian polynomial matrix to transform it to a diagonal polynomial matrix and can therefore be thought of as an extension

[1]Note that a paraunitary similarity transformation matrix represents the polynomial equivalent to the unitary similarity transformations discussed in [6].

of the conventional Jacobi algorithm suitable for scalar Hermitian matrices [6]. In fact, if the algorithm is applied to a scalar Hermitian matrix, it simply reduces to diagonalising the matrix via Jacobi's algorithm. Although polynomial matrices and the idea of polynomial matrix decompositions are commonly established ideas in signal processing, as discussed in Chapter 2 and [16], the only previous attention to formulating a routine suitable for calculating an EVD or SVD of a polynomial matrix is that of Lambert [20,52,53]. However, the SBR2 algorithm operates entirely in the time-domain and is therefore very different from Lambert's EVD routine, which involves the approximate inversion of filters in the frequency domain.

This chapter firstly describes how the SBR2 algorithm operates. Convergence of the algorithm is proven and its application to strong decorrelation and subspace decomposition is highlighted by means of a simple numerical example. Other applications of the algorithm include broadband adaptive beamforming and the design of filterbanks for optimal data compaction, [10]. The SBR2 algorithm has since been applied to other areas of signal processing, such as MIMO communications where it can be used to decouple a MIMO communication system into a set of independent subchannels, [13,14,55]. This along with other applications of the decomposition are discussed in the penultimate chapter of this thesis.

3.2 The Eigenvalue Decomposition of a Polynomial Matrix

The eigenvalue decomposition of a para-Hermitian polynomial matrix, $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times p}$, is defined here as

$$\underline{\mathbf{R}}(z) = \underline{\widetilde{\mathbf{H}}}(z)\underline{\mathbf{D}}(z)\underline{\mathbf{H}}(z) \tag{3.1}$$

where the polynomial matrix $\underline{\mathbf{H}}(z) \in \underline{\mathbb{C}}^{p\times p}$ is paraunitary and the resulting polynomial matrix $\underline{\mathbf{D}}(z) \in \underline{\mathbb{C}}^{p\times p}$ is diagonal. Note that the matrix to be diagonalised $\underline{\mathbf{R}}(z)$ must be para-Hermitian, which means all coefficients of the matrix must satisfy $r_{jk}(\tau) = r^*_{kj}(-\tau)$ for $j,k = 1,\ldots,p$ and for all values of the lag parameter $\tau \in \mathbb{Z}$.

The SBR2 algorithm can be used to calculate the paraunitary matrix $\underline{\mathbf{H}}(z)$ suitable for transforming the para-Hermitian input matrix $\underline{\mathbf{R}}(z)$, into the diagonal matrix $\underline{\mathbf{D}}(z)$ according to equation (3.1). As the matrix $\underline{\mathbf{H}}(z)$ is paraunitary, the transformation will be energy preserving, which means the Frobenius norm of the two matrices $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{D}}(z)$ are equal, i.e.

$$\|\underline{\mathbf{R}}(z)\|_F = \|\underline{\mathbf{D}}(z)\|_F . \tag{3.2}$$

3.3 The Sequential Best Rotation Algorithm

Given the para-Hermitian polynomial matrix $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times p}$, the objective of the SBR2 algorithm is to compute the paraunitary polynomial matrix $\underline{\mathbf{H}}(z) \in \underline{\mathbb{C}}^{p\times p}$ such that

$$\underline{\mathbf{H}}(z)\underline{\mathbf{R}}(z)\widetilde{\underline{\mathbf{H}}}(z) = \underline{\mathbf{D}}(z), \tag{3.3}$$

where the polynomial matrix $\underline{\mathbf{D}}(z) \in \underline{\mathbb{C}}^{p\times p}$ is diagonal. The algorithm operates as an iterative process, where at each iteration an elementary paraunitary transformation is applied to both sides of the polynomial matrix $\underline{\mathbf{R}}(z)$ designed to drive the two off-diagonal coefficients with the largest magnitude to zero. The paraunitary polynomial transformation matrix $\underline{\mathbf{H}}(z)$ is therefore formulated as a series of elementary paraunitary matrices, i.e.

$$\underline{\mathbf{H}}(z) = \underline{\mathbf{G}}_i(z) \ldots \underline{\mathbf{G}}_1(z), \tag{3.4}$$

where i denotes the unspecified number of iterations required to diagonalise the matrix and $\underline{\mathbf{G}}_i(z)$ is the elementary paraunitary matrix calculated at iteration i. Each of these matrices consists of a complex elementary scalar rotation matrix, $\mathbf{Q}^{(j,k)}(\theta,\phi)$, preceded by an elementary delay matrix, $\underline{\mathbf{B}}^{(k,t)}(z)$ and is formulated as

$$\underline{\mathbf{G}}_i(z) = \mathbf{Q}^{(j,k)}(\theta,\phi)\underline{\mathbf{B}}^{(k,t)}(z) \tag{3.5}$$

where at each iteration the parameters j, k, t, θ and ϕ are appropriately chosen depending on

the coefficients within the matrix $\underline{\mathbf{R}}(z)$ that the elementary paraunitary matrix is attempting to annihilate. The matrix from equation (3.4) will clearly be paraunitary as each term in equation (3.5) is paraunitary. The two types of elementary paraunitary matrices will now be discussed.

3.3.1 An Elementary Rotation Matrix

The elementary scalar rotation matrix $\mathbf{Q}^{(j,k)}(\theta,\phi)$ takes the form of a $p \times p$ identity matrix with the exception of the four elements positioned at the intersection of rows j and k with columns j and k. These elements are given by the elements of the 2×2 submatrix $\widehat{\mathbf{Q}}^{(j,k)}(\theta,\phi)$, which is formulated as

$$\widehat{\mathbf{Q}}^{(j,k)}(\theta,\phi) = \begin{bmatrix} c & se^{i\phi} \\ -se^{-i\phi} & c \end{bmatrix}, \tag{3.6}$$

where c and s define respectively the cosine and sine of the angle θ in radians. The angles θ and ϕ are chosen, so that when the matrix $\mathbf{Q}^{(j,k)}(\theta,\phi)$ is applied to a polynomial matrix $\underline{\mathbf{A}}(z)$ as follows

$$\underline{\mathbf{A}}'(z) = \mathbf{Q}^{(j,k)}(\theta,\phi)\underline{\mathbf{A}}(z) \tag{3.7}$$

one coefficient from the polynomial element $\underline{a}'_{jk}(z)$ is rotated to equal zero. This will of course affect all coefficients associated with polynomial elements in the j^{th} and k^{th} rows of the matrix to which it is applied. Similarly, post-multiplication of $\underline{\mathbf{A}}(z)$ by the para-conjugate of $\mathbf{Q}^{(j,k)}$ will similarly rotate all coefficients in the j^{th} and k^{th} columns of the polynomial matrix. Again appropriate choices of the rotation angles θ and ϕ can result in driving a single coefficient in the $(k,j)^{th}$ polynomial element of the matrix to zero. If the matrix $\underline{\mathbf{A}}(z)$ is para-Hermitian, then this process can be implemented to drive two coefficients of an off-diagonal polynomial to zero, but will require two rotations, one from the left and one from the right, using the same angles.

3.3.2 An Elementary Delay Matrix

The second component of the elementary paraunitary transformation matrix is an elementary delay matrix, $\underline{\mathbf{B}}^{(k,t)}(z) \in \underline{\mathbb{C}}^{p\times p}$. This matrix takes the form of a $p \times p$ identity matrix with the exception of the k^{th} diagonal element, which is z^{-t}, i.e.

$$\underline{\mathbf{B}}^{(k,t)}(z) = \begin{bmatrix} \mathbf{I}_{k-1} & 0 & 0 \\ 0 & z^{-t} & 0 \\ 0 & 0 & \mathbf{I}_{p-k} \end{bmatrix}. \tag{3.8}$$

The objective of this matrix is to impose a delay of size t to all elements in the k^{th} row of the polynomial matrix to which it is applied. All other rows of the matrix are unaffected by this transformation.

The SBR2 algorithm can now be broken down into a three-step iterative process using these two elementary matrices, which will now be described in detail. Note that the algorithm is also described in detail in [7].

3.3.3 The SBR2 Algorithm

The algorithm begins each iteration by locating the coefficient associated with an off-diagonal polynomial element within $\underline{\mathbf{R}}(z)$ with the largest magnitude and so the objective is to find the coefficient $r_{jk}(t)$, where $j \neq k$, such that

$$|r_{jk}(t)| \geq |r_{lm}(\tau)| \tag{3.9}$$

holds for all coefficients $r_{lm}(\tau)$ in $\underline{\mathbf{R}}(z)$ where $l, m = 1, \ldots, p$ with $l \neq m$ and $\tau \in \mathbb{Z}$. This coefficient will be referred to as the dominant coefficient and its absolute value is defined as

$$g = |r_{jk}(t)|. \tag{3.10}$$

Suppose the dominant coefficient is found to be $r_{jk}(t)$, i.e. the coefficient of z^{-t} in the $(j,k)^{th}$ polynomial element, then this coefficient will be repeated as the coefficient $r^*_{kj}(-t)$ due to

the structure of the para-Hermitian matrix defined in Section 2.4.2. For this reason the search for the dominant coefficient can be restricted to only the elements above the diagonal of $\underline{\mathbf{R}}(z)$ and so we require $j < k$ and $l < m$ in equation (3.9). If the dominant coefficient is not unique then any of the dominant coefficients may be chosen. Note that the specific parameters j, k and t define the position of the dominant coefficient within the polynomial matrix and are now used to formulate the elementary paraunitary matrices required to drive the two dominant coefficients to zero.

Having found the dominant off-diagonal coefficient and its associated indices j, k and t, the second step of the algorithm is to apply the appropriate delay matrix to $\underline{\mathbf{R}}(z)$ to obtain the transformed matrix

$$\underline{\mathbf{R}}'(z) = \underline{\mathbf{B}}^{(k,t)}(z)\underline{\mathbf{R}}(z)\widetilde{\underline{\mathbf{B}}}^{(k,t)}(z). \tag{3.11}$$

Subsequently, all coefficients in the k^{th} row and k^{th} column of the matrix, excluding the diagonal coefficients, have been shifted such that the dominant coefficient pair are now the coefficients of z^0, i.e. $r'_{jk}(0) = r_{jk}(t) = r'_{kj}(-t)^* = r'_{kj}(0)^*$. Note that to accommodate all of the shifted coefficients the order of the matrix will have grown by $2\,|t|$.

The third and final step of each iteration of the SBR2 algorithm is to apply the appropriate elementary rotation matrix to $\underline{\mathbf{R}}'(z)$ to obtain

$$\underline{\mathbf{R}}''(z) = \mathbf{Q}^{(j,k)}(\theta,\phi)\underline{\mathbf{R}}'(z)(\mathbf{Q}^{(j,k)}(\theta,\phi))^H. \tag{3.12}$$

The rotation angles θ and ϕ required to drive the dominant coefficient to zero are chosen such that

$$\begin{bmatrix} c & se^{i\phi} \\ -se^{-i\phi} & c \end{bmatrix}\begin{bmatrix} r'_{jj}(0) & r'_{jk}(0) \\ r'_{kj}(0) & r'_{kk}(0) \end{bmatrix}\begin{bmatrix} c & -se^{i\phi} \\ se^{-i\phi} & c \end{bmatrix} = \begin{bmatrix} r''_{jj}(0) & 0 \\ 0 & r''_{kk}(0) \end{bmatrix}. \tag{3.13}$$

This condition is satisfied when the angles are calculated such that

$$\phi = arg(r'_{jk}(0)) \tag{3.14}$$

and

$$\tan(2\theta) = \frac{2\left|r'_{jk}(0)\right|}{r'_{jj}(0) - r'_{kk}(0)}. \tag{3.15}$$

Equation (3.15) has multiple solutions for θ, any of which can be used. If the basic inverse tangent function (arctangent) is used to calculate θ it will produce a solution in the range $(-\pi/4, \pi/4]$. However, it is preferable to use the four quadrant inverse tangent function as it will produce a $\theta \in (-\pi/2, \pi/2]$, which typically leads to output channels that are ordered in terms of decreasing power.

The application of the rotation matrix, as shown in equation (3.13), will not only affect the two dominant coefficients, but will also change all coefficients in rows and columns j and k of $\underline{\mathbf{R}}'(z)$. The application of the rotation matrix will clearly not alter the order of the matrix.

This completes the first iteration of the SBR2 algorithm, resulting in an overall transformation of the form

$$\underline{\mathbf{G}}_1(z)\underline{\mathbf{R}}(z)\widetilde{\underline{\mathbf{G}}}_1(z) = \underline{\mathbf{D}}_1(z), \tag{3.16}$$

where $\underline{\mathbf{G}}_1(z) = \mathbf{Q}^{(j,k)}(\theta, \phi)\underline{\mathbf{B}}^{(k,t)}(z)$ and $\underline{\mathbf{D}}_1(z)$ is the polynomial matrix resulting from the transformation with coefficients $d_{jk}(0) = d_{kj}(0) = 0$.

Each element of the matrix $\underline{\mathbf{R}}(z)$ involves a number of polynomial coefficients and so in practice it will not be possible to zero all coefficients of every off-diagonal polynomial element and achieve exact digaonalisation. Instead the algorithm continues until all off-diagonal coefficients of $\underline{\mathbf{D}}(z)$ are sufficiently small and the magnitude of all off-diagonal coefficients satisfy

$$|r_{jk}(t)| < \epsilon \tag{3.17}$$

for $j, k = 1, \ldots, p$ with $j \neq k$, $t \in \mathbb{Z}$ and where $\epsilon > 0$ is a prespecified small value. Although exact diagonalisation is not feasible, a good approximation can be achieved and this is exemplified by the numerical example in Section 3.7.

The algorithm repeats this three step process iteratively, replacing $\underline{\mathbf{R}}(z)$ with $\underline{\mathbf{R}}''(z)$ until the transformed polynomial matrix is sufficiently diagonal according to equation (3.17). Com-

pleting a number of iterations of the SBR2 algorithm, say N, will result in a transformation of the form

$$\underline{\mathbf{H}}_N(z)\underline{\mathbf{R}}(z)\underline{\widetilde{\mathbf{H}}}_N(z) = \underline{\mathbf{D}}_N(z), \tag{3.18}$$

where $\underline{\mathbf{H}}_N(z)$ is the overall transformation matrix after N iterations and consists of N elementary paraunitary matrices of the form shown in equation (3.5). The matrix $\underline{\mathbf{D}}_N(z)$ is the output of the algorithm after N iterations and converges to the diagonal matrix $\underline{\mathbf{D}}(z)$ as the number of iterations increases. Note also that the matrix $\underline{\mathbf{H}}_N(z)$ is paraunitary as it consists of the product of N paraunitary matrices. Furthermore the transformation is norm preserving and so

$$\left\|\underline{\mathbf{H}}_N(z)\underline{\mathbf{R}}(z)\underline{\widetilde{\mathbf{H}}}_N(z)\right\|_F = \|\underline{\mathbf{R}}(z)\|_F\,. \tag{3.19}$$

3.4 Convergence of the SBR2 Algorithm

To discuss convergence of the algorithm, four measures are introduced; firstly N_1 the squared Frobenius norm of the diagonal elements on the coefficient matrix of z^0, i.e. $\mathbf{R}(0)$, N_2 the squared Frobenius norm of the coefficient matrix of z^0, N_3 the squared Frobenius norm of the off-diagonal coefficients of z^0 and finally N_4 the squared Frobenius norm for the matrix $\underline{\mathbf{R}}(z)$ [7]. These measures can be calculated as

$$N_1 = \sum_{j=1}^{p} |r_{jj}(0)|^2\,, \tag{3.20}$$

$$N_2 = \sum_{j=1}^{p}\sum_{k=1}^{p} |r_{jk}(0)|^2\,, \tag{3.21}$$

$$N_3 = \sum_{j=1}^{p}\sum_{\substack{k=1\\k\neq j}}^{p} |r_{jk}(0)|^2 \tag{3.22}$$

and

$$N_4 = \sum_{\tau}\sum_{j=1}^{p}\sum_{k=1}^{p} |r_{jk}(\tau)|^2\,. \tag{3.23}$$

Note that the quantity N_1 is not affected by applying the elementary delay matrices and

remains constant. However the quantities N_2 and N_3 will be affected whenever the parameter t in equation (3.11) does not equal zero. Following the rotation step at each iteration the quantity N_3 will decrease by the magnitude squared of the two dominant coefficients. Furthermore the relationship between the first three quantities specified by equations (3.20)-(3.22) can be expressed as

$$N_2 = N_1 + N_3, \tag{3.24}$$

which will remain constant through the rotation step and so the quantity N_1 will increase relating to the decrease in N_3.

Therefore, at each iteration N_1 will increase by twice the magnitude squared of the dominant coefficient, i.e. the quantity $2g^2$, and so will increase monotonically over the series of iterations. N_4 will not be affected by any application of a delay matrix or rotation matrix, as the transformation is norm preserving, and so will remain constant over all iterations of the algorithm. Since N_1 increases monotonically and is bounded above by N_4 the total energy in the matrix, which is constant, it must have a supremum S. As a result, there must be an iteration L for which $|S - N_1| < \epsilon$ for any $\epsilon > 0$. Then at any subsequent stage the same quantity must satisfy $2g^2 < |S - N_1| < \epsilon$ and so there must be an iteration by which the magnitude squared of the dominant coefficient at that iteration is bounded by ϵ. Hence the stopping condition specified by equation (3.17) can be guaranteed and convergence of the SBR2 algorithm confirmed.

3.5 Implementation and Computational Complexity of the SBR2 Algorithm

The algorithm is designed to stop when either a set number of iterations is completed or when the stopping criterion specified by equation (3.17) is met. Currently, this condition is specified in terms of the smallness of the magnitude of the maximum off-diagonal coefficient, g, in relation to the quantity $\sqrt{N_1/p}$, which represents a lower bound for the maximum autocorrelation value at lag zero. The algorithm is therefore set to stop when the following

condition is satisfied

$$g < \epsilon = \delta\sqrt{N_1/p}, \tag{3.25}$$

where $\delta > 0$ is a prespecified small value and p denotes the number of number of rows in the para-Hermitian input matrix $\underline{\mathbf{R}}(z)$, which for application purposes is equal to the number of received signals.

To reduce the computational time of SBR2, the paraunitary matrix $\underline{\mathbf{H}}(z)$ is not stored or calculated within the algorithm. Instead the series of parameters j, k, t, θ and ϕ required for computing $\underline{\mathbf{H}}(z)$ are stored and then the matrix can be computed, if required, once all iterations of the algorithm have been completed. Also the rotation or delay matrices are not applied to the entire polynomial matrix when implementing the algorithm as this would increase the computational complexity of the algorithm unnecessarily. Only two rows and columns are affected by the rotation and one row and column by the application of the delay matrix and so only these rows and columns are changed. The number of computations calculated at each iteration can be further reduced by exploiting the para-conjugate symmetry of the para-Hermitian matrix $\underline{\mathbf{R}}(z)$.

The computational complexity of the SBR2 algorithm is discussed in Appendix C. It is very much dependent upon the size and order of the updated polynomial matrix at iteration i, $\underline{\mathbf{D}}_i(z)$. However, it is generally unnecessarily high due to the algorithm storing all coefficients of the polynomial matrix, even if coefficients associated with the outer time lags of the matrix are very small or equal to zero. This problem is highlighted in example 3.7 and Chapter 4 discusses techniques of alleviating this problem.

3.6 Applications of the SBR2 Algorithm

3.6.1 Strong Decorrelation

Chapter 2 discussed the conventional EVD for scalar matrices as a technique for decorrelating a set of instantaneously mixed signals, where there are no time delays in the propagation of the signals from the sources to the sensors. In this case the signals received at the sensors

can be decorrelated by applying a unitary matrix that modifies the signals in both phase and amplitude and this matrix can be obtained by calculating the EVD of the sample covariance matrix of the received signals.

However, this method cannot be applied to a set of convolutively mixed signals, which will not only be correlated with each other at the same time instant, but possibly over a range of time delays as well. Instead a polynomial matrix of chosen filters will be required to transform the received signals and impose strong decorrelation, i.e. decorrelation of the signals at the same time instant and over any relative time delay. The matrix required to impose strong decorrelation must preserve the total energy in the signals through the transformation and can be found by calculating the EVD of the polynomial space-time covariance matrix for the convolutively mixed signals using the SBR2 algorithm.

Firstly the convolutive mixing model is defined, before discussing in detail how strong decorrelation of a set of convolutively mixed signals is achieved. It is assumed that a set of source signals $\mathbf{s}(t) \in \mathbb{C}^{q\times 1}$ for $t \in \{0, 1, \ldots, T-1\}$ are emitted from q independent sources through a convolutive channel, to be received at an array of p sensors, where it is assumed that $p \geq q$. The mixing model for the set of convolutively mixed signals, $\mathbf{x}(t) \in \mathbb{C}^p$ where $t \in \{0, 1, \ldots, T-1\}$ can be expressed as

$$\mathbf{x}(t) = \sum_{k=0}^{N} \mathbf{C}(k)\mathbf{s}(t-k) + \mathbf{n}(t) \tag{3.26}$$

where $\underline{\mathbf{C}}(z) = \sum_{k=0}^{N} \mathbf{C}(k)z^{-k}$ denotes the polynomial mixing matrix with coefficient matrices $\mathbf{C}(k) \in \mathbb{C}^{p\times q}$ for $k = \{0, 1, \ldots, N\}$ and $\mathbf{n}(t) \in \mathbb{C}^p$ denotes an additive zero-mean noise process with variance $\sigma^2\mathbf{I}$. In the above expression N defines the order of the mixing matrix. This mixing model can alternatively be written as

$$\underline{\mathbf{x}}(z) = \underline{\mathbf{C}}(z)\underline{\mathbf{s}}(z) + \underline{\mathbf{n}}(z), \tag{3.27}$$

where $\underline{\mathbf{x}}(z)$, $\underline{\mathbf{s}}(z)$ and $\underline{\mathbf{n}}(z)$ define algebraic power series of the form

$$\underline{\mathbf{x}}(z) = \sum_{t=-\infty}^{\infty} \mathbf{x}(t) z^{-t} \tag{3.28}$$

of the received signals, the source signals and the additive noise terms respectively.

Assuming that the received signals generated by mixing model (3.26) have zero mean then the space-time covariance matrix for the set of signals is defined as

$$\underline{\mathbf{R}}_{xx}(z) = \sum_{\tau=-\infty}^{\infty} \mathbf{R}_{xx}(\tau) z^{-\tau} \tag{3.29}$$

where $\mathbf{R}_{xx}(\tau) = \mathrm{E}\left[\mathbf{x}(t)\mathbf{x}^H(t-\tau)\right]$ for $\tau \in \mathbb{Z}$. However, in practice the estimated space-time covariance matrix for the signals is calculated as

$$\underline{\widehat{\mathbf{R}}}_{xx}(z) = \sum_{\tau=-W}^{W} \widehat{\mathbf{R}}_{xx}(\tau) z^{-\tau} \tag{3.30}$$

where

$$\widehat{\mathbf{R}}_{xx}(\tau) = \sum_{t=0}^{T-1} \frac{\mathbf{x}(t)\mathbf{x}^H(t-\tau)}{T} \tag{3.31}$$

and W defines the correlation lag window parameter. This parameter can be chosen experimentally or, if appropriate knowledge of the mixing system is available, an informed choice can be made. The space-time covariance matrix $\underline{\widehat{\mathbf{R}}}_{xx}(z)$ is unlikely to be diagonal as the received signals will generally be correlated with one another. In calculating this matrix, the following three assumptions are made

1. $T >> W$,
2. $\mathbf{x}(t) = 0$ outside of the sample interval $[0, 1, \ldots, T-1]$ and
3. $\mathbf{R}(\tau) = 0$ for $|\tau| > W$.

The order of this estimated space-time covariance matrix is $2W$. It is easily demonstrated that this matrix is para-Hermitian as the auto and cross-correlation sequences of the received

signals, for example $x_i(t)$ and $x_j(t)$, will satisfy

$$r_{x_i x_j}(\tau) = r^*_{x_j x_i}(-\tau) \tag{3.32}$$

for $\tau \in \mathbb{Z}$ and so $\widehat{\underline{\mathbf{R}}}(z) = \widetilde{\underline{\mathbf{R}}}(z)$ enabling the matrix to be a suitable input to the SBR2 algorithm. The SBR2 algorithm when applied to a space-time covariance matrix can be used to obtain a polynomial EVD (PEVD) of $\underline{\mathbf{R}}(z)$ and therefore to find the polynomial paraunitary matrix required to enforce strong decorrelation upon the set of received signals. Applying SBR2 to the estimated space-time covariance matrix, $\widehat{\underline{\mathbf{R}}}_{xx}(z)$, produces the decomposition

$$\underline{\mathbf{H}}(z)\widehat{\underline{\mathbf{R}}}_{xx}(z)\widetilde{\underline{\mathbf{H}}}(z) = \widehat{\underline{\mathbf{D}}}(z) \tag{3.33}$$

where $\underline{\mathbf{H}}(z)$ is the paraunitary transformation matrix and $\widehat{\underline{\mathbf{D}}}(z)$ is approximately diagonal. The transformed signals can then be calculated as

$$\underline{\mathbf{y}}(z) = \underline{\mathbf{H}}(z)\underline{\mathbf{x}}(z), \tag{3.34}$$

which, to a good approximation, are strongly decorrelated. The space-time covariance matrix for the signals can be estimated by

$$\widehat{\underline{\mathbf{R}}}_{yy}(z) = \underline{\mathbf{H}}(z)\widehat{\underline{\mathbf{R}}}_{xx}(z)\widetilde{\underline{\mathbf{H}}}(z) = \widehat{\underline{\mathbf{D}}}(z), \tag{3.35}$$

which is an approximately diagonal polynomial matrix. The process of strongly decorrelating a set of convolutively mixed signals by the SBR2 algorithm is demonstrated by a simple numerical example in Section 3.7.

One clear advantage of diagonalising the polynomial matrix by means of paraunitary transformations, such as that obtained by the SBR2 algorithm, is that the transformation is norm preserving, i.e. $\|\underline{\mathbf{D}}(z)\|_F^2 = \left\|\widehat{\underline{\mathbf{R}}}_{xx}(z)\right\|_F^2$, and so no information is lost over the series of iterations of the algorithm. Also the transformation will not amplify any additive noise terms as the variance of the noise following the paraunitary transformation will remain constant.

3.6.2 Properties of the SBR2 Algorithm

This section aims to briefly discuss the other properties or applications of the SBR2 algorithm.

1. As discussed previously the paraunitary transformation carried out by the SBR2 algorithm is norm preserving and so $\left\|\underline{\mathbf{H}}(z)\underline{\mathbf{R}}(z)\widetilde{\underline{\mathbf{H}}}(z)\right\|_F^2 = \|\underline{\mathbf{D}}(z)\|_F^2$.

2. Two signals are said to be spectrally majorised if the expected power in one signal is greater that the expected power of the other signal at every frequency. After applying the SBR2 algorithm to the space-time covariance matrix for a set of convolved signals, the output signals are ordered in decreasing magnitude of total energy such that

$$d_{11}(0) \geq d_{22}(0) \geq \ldots \geq d_{pp}(0). \tag{3.36}$$

   In addition, the SBR2 algorithm has a tendency to produce spectrally majorised outputs, although this is not always a guaranteed property of the decomposition. This property will be demonstrated by the example in Section 3.7.

3. The signal and noise subspaces can be identified using the SBR2 algorithm. This is performed by inspection of the power spectral density (PSD) of the decorrelated signals, which can also be used for identifying if the signals are spectrally majorised. Section 3.6.3 will now briefly discuss how the PSD of the received and decorrelated signal can be calculated.

3.6.3 Power Spectrum of the Signals

The power spectrum of a signal describes the distribution of power with frequency, demonstrating which frequencies are present and how much power each frequency possesses. For a wide-sense stationary (WSS) process it is easily calculated, according to the Wiener-Khinchin Theorem, by taking the Fourier Transform of the autocorrelation sequence, $r_{xx}(n)$, for the signal as follows

$$S_{xx}(\omega) = \sum_{n=-\infty}^{\infty} r_{xx}(n)e^{-j\omega n} \tag{3.37}$$

where ω represents normalised angular frequency [56]. The estimated autocorrelation sequences for the set of received signals correspond to the diagonal elements of the space-time covariance matrix $\underline{\widehat{\mathbf{R}}}_{xx}(z)$. Alternatively, the expected sequences correspond to the diagonal elements of the matrix

$$\underline{\mathbf{R}}_{xx}(z) = \underline{\mathbf{C}}(z)\underline{\mathbf{R}}_{ss}(z)\widetilde{\underline{\mathbf{C}}}(z) + \sigma^2\mathbf{I}_p \tag{3.38}$$

where σ^2 defines the variance of the noise and $\underline{\mathbf{R}}_{ss}(z)$ defines the expected polynomial space-time covariance matrix of the source signals, which as the source signals are assumed to be statistically independent will be diagonal. Furthermore, if the source signals have unit variance, such as those used in example 3.7, $\underline{\mathbf{R}}_{ss}(z) = \mathbf{I}_q$.

The estimated autocorrelation sequences of the output signals are the diagonal elements of the diagonalised spectral density matrix $\underline{\mathbf{D}}(z)$ obtained by the SBR2 algorithm. Alternatively, the expected sequences could have been calculated, using the similarity transformation matrix, $\underline{\mathbf{H}}(z)$, and the mixing matrix, $\underline{\mathbf{C}}(z)$, by computing

$$\underline{\mathbf{R}}_{yy}(z) = \underline{\mathbf{H}}(z)\underline{\mathbf{C}}(z)\widetilde{\underline{\mathbf{C}}}(z)\widetilde{\underline{\mathbf{H}}}(z) + \sigma^2\mathbf{I}_p. \tag{3.39}$$

Evaluating the diagonal entries of $\underline{\mathbf{R}}_{yy}(z)$ will yield the expected autocovariance functions of the decorrelated signals $\mathbf{y}(t)$.

The algorithm can also be used to identify the signal and noise subspaces. From inspecting plots of the PSD the noise signals can be identified as the channels with low spectra and therefore positioned at the bottom of the graph. The remaining signals, each with considerably larger spectra, correspond to the decorrelated signals.

3.7 Numerical Example

A simple example is given to demonstrate how the SBR2 algorithm operates and briefly illustrate a couple of applications to signal processing, specifically using the SBR2 algorithm as a technique for enforcing strong decorrelation upon a set of convolved signals and also as a tool for subspace decomposition. A polynomial mixing matrix $\underline{\mathbf{C}}(z) \in \underline{\mathbb{C}}^{4\times 3}$ was generated,

specifically designed to emulate the propagation of three signals onto four sensors. Each of the polynomial elements of the matrix was chosen to be a fifth order FIR filter, where the coefficients were drawn from a uniform distribution in the range $[-1, 1]$.

Three independent BPSK source signals, each of length 1000, were then generated and convolutively mixed according to equation (3.26), where N defines the order of the mixing matrix and is, in this case, equal to five. The variance of the noise process was chosen to give a signal-to-noise ratio (SNR) at the receiver of 5 dB. The space-time covariance matrix for the system could then be calculated according to equations (3.30) and (3.31) where the correlation window parameter was set as $W = 10$. As each element of the mixing matrix $\underline{\mathbf{C}}(z)$ is a fifth order FIR filter, the estimated auto and cross-correlation sequences for each of the received signals will be approximately zero for all lags $|\tau| > 5$ and any deviation in these areas will be due to sample estimation errors. For this reason the choice of our lag window parameter W is more than sufficient, in fact $W = 5$ would have been adequate. A graphical representation of the estimated polynomial space-time covariance matrix $\underline{\widehat{\mathbf{R}}}(z)$ can be seen in Figure 3.1, where stem plots are used to illustrate the series of coefficients for each of the polynomial elements. The position of the stem plot in the figure corresponds to the position of the polynomial element, which it represents within the matrix. In this example, both the source signals and the polynomial elements of the mixing matrix were chosen to be real, resulting in the space-time covariance matrix also consisting of polynomial elements with real coefficients. The algorithm could have equally been applied to a polynomial matrix whose coefficients are complex.

The SBR2 algorithm when applied to this matrix, took 184 iterations to converge to a point where g, the off-diagonal coefficient with maximum magnitude in $\underline{\mathbf{D}}(z)$, is less than $\sqrt{\frac{N_1}{4}} \times 10^{-2}$. This is demonstrated in Figure 3.2, where the magnitude of the dominant coefficient at each iteration is plotted over the series of iterations.

The order of the matrices $\underline{\mathbf{D}}(z)$ and $\underline{\mathbf{H}}(z)$ following all 182 iterations of the algorithm are 3486 and 1733 respectively. The input matrix $\underline{\widehat{\mathbf{R}}}(z)$ has a squared Frobenius norm of 478.19, with 173.38 positioned in the diagonal elements. This accounts for 36.18% of the total squared Frobenius norm of the matrix. However, following the application of the SBR2 algorithm,

the squared Frobenius norm of the off-diagonal elements decreased to 1.32, amounting to 0.28% of the total squared Frobenius norm of the matrix. The diagonal matrix $\underline{\mathbf{D}}(z)$ and the paraunitary transformation matrix $\underline{\mathbf{H}}(z)$ can be seen in Figures 3.3 and 3.4 respectively. Upon inspection of the plots of these matrices the order of these matrices can be seen to be unnecessarily large, with the majority of the coefficients positioned in the outer lags amounting to a small proportion of the Frobenius norm of the polynomial matrix. This will be discussed further in Section 3.9. In fact the outer 3400 coefficient matrices of $\underline{\mathbf{D}}(z)$ account for only 0.0065% of the total squared Frobenius norm of the matrix.

Figures 3.5 and 3.6 demonstrate the power spectral density (PSD) of the mixed signals and the decorrelated signals, following the application of the paraunitary transformation matrix obtained from the SBR2 algorithm, respectively. The PSDs were calculated according to Section 3.6.3. From Figure 3.6 it can be seen that approximate spectral majorisation has been achieved and the noise term can be identified as the signal with the lowest spectra, i.e. output four. The signal and noise subspaces have thus been identified.

A parunitary transformation, such as that carried out by the SBR2 algorithm, is norm preserving and therefore the total energy of all signals throughout the transformation remains constant. Furthermore, the transformation preserves the total energy at each frequency, [16, 57, 58], and this can be seen in Figure 3.7, where the total PSD in all signals is plotted before and after applying the SBR2 algorithm. Note that the transformation can redistribute the power between channels, but it cannot allow the total power to increase or decrease.

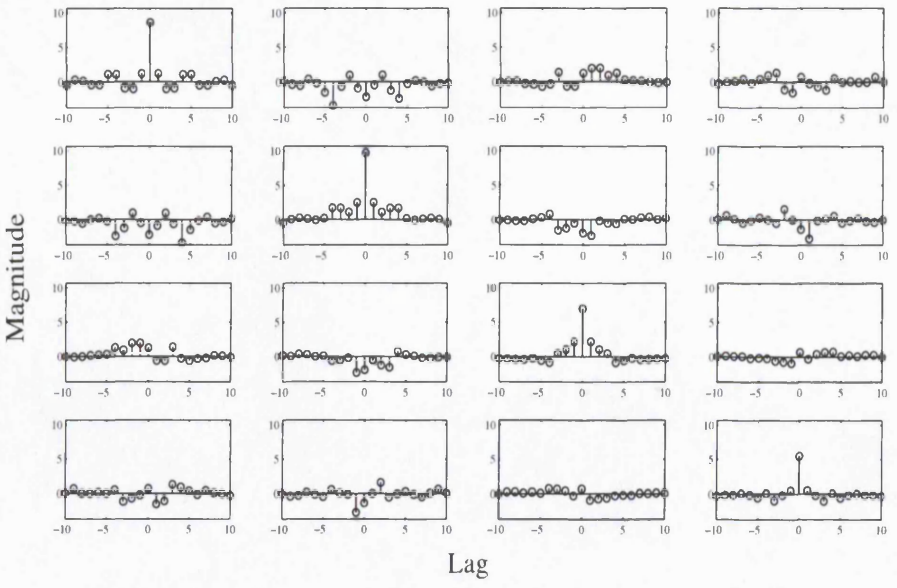

Figure 3.1: A stem plot representation of the para-Hermitian polynomial space-time covariance matrix $\underline{\widehat{\mathbf{R}}}(z)$ to be used as input to the SBR2 algorithm.

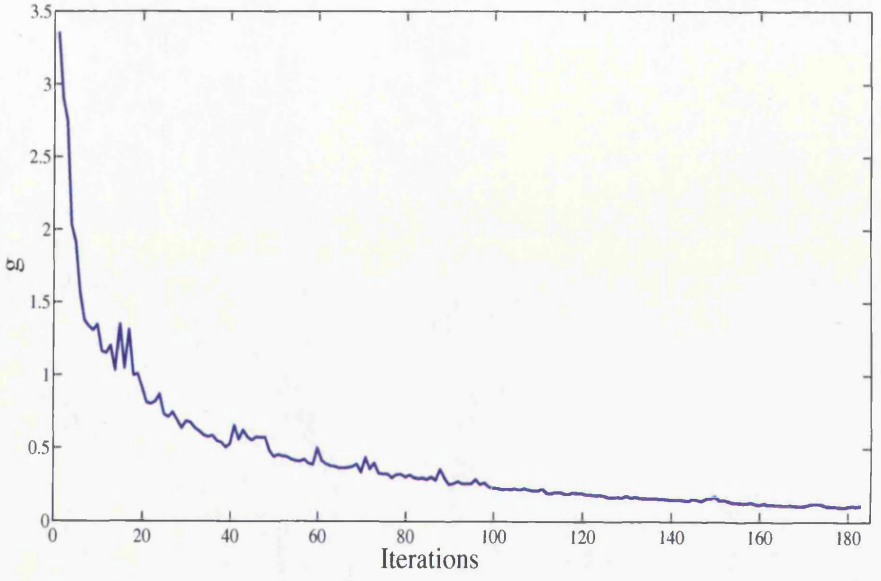

Figure 3.2: The magnitude of the dominant coefficient, g, over the series of iterations.

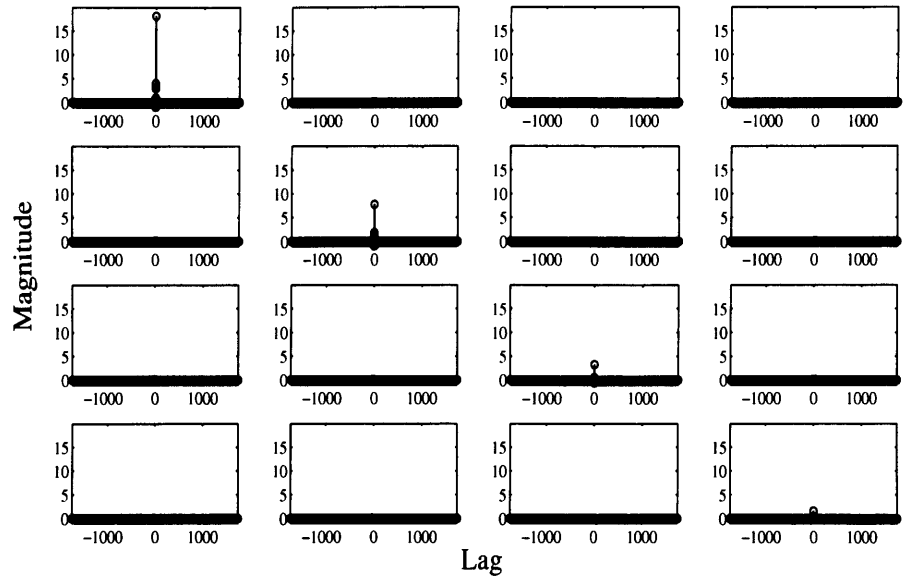

Figure 3.3: A stem plot representation of the diagonalised polynomial matrix $\underline{\mathbf{D}}(z)$ obtained when the SBR2 algorithm is applied to the polynomial matrix $\underline{\widehat{\mathbf{R}}}(z)$ demonstrated in Figure 3.1.

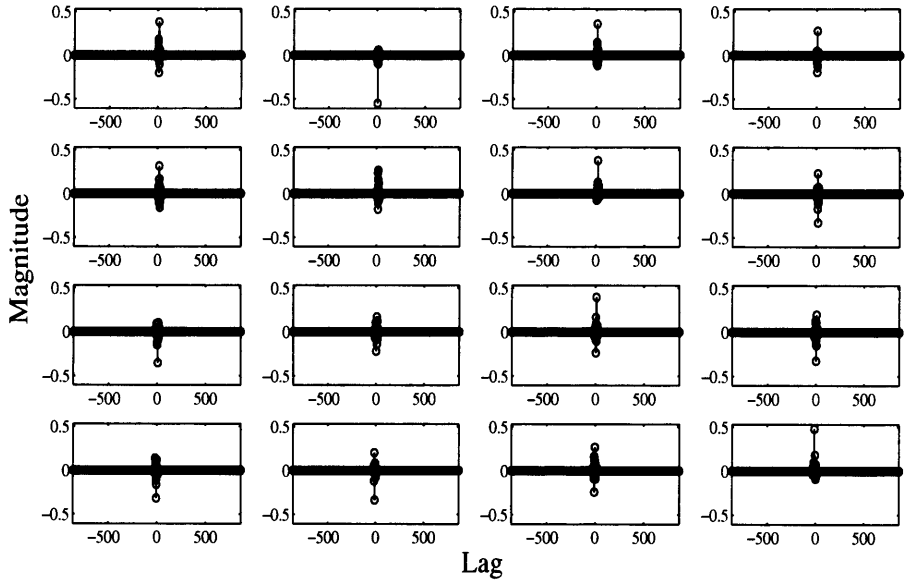

Figure 3.4: A stem plot representation of the paraunitary polynomial matrix $\underline{\mathbf{H}}(z)$ obtained when the SBR2 algorithm is applied to the matrix $\underline{\widehat{\mathbf{R}}}(z)$ demonstrated in Figure 3.1.

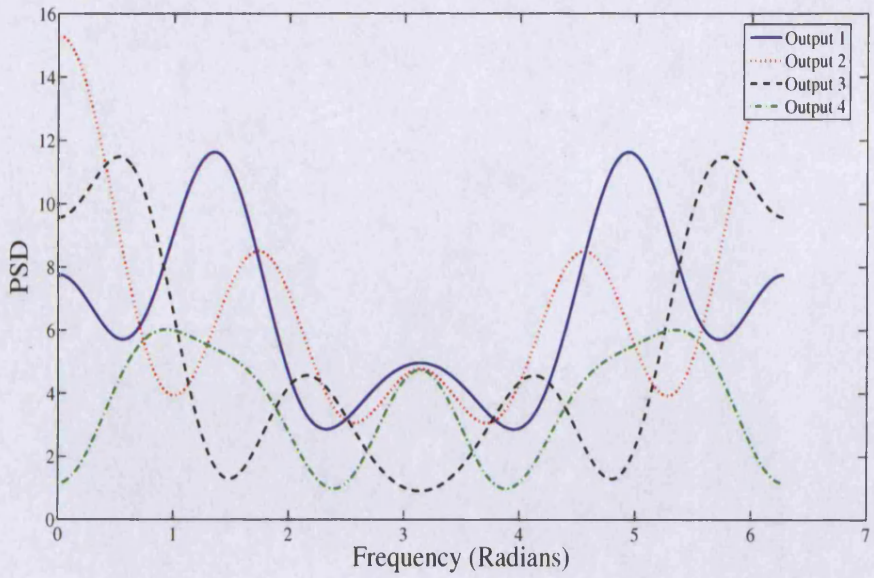

Figure 3.5: Plot of the spectra of the convolutively mixed signals, whose space-time covariance matrix $\underline{\mathbf{R}}(z)$ was used as input to the SBR2 algorithm.

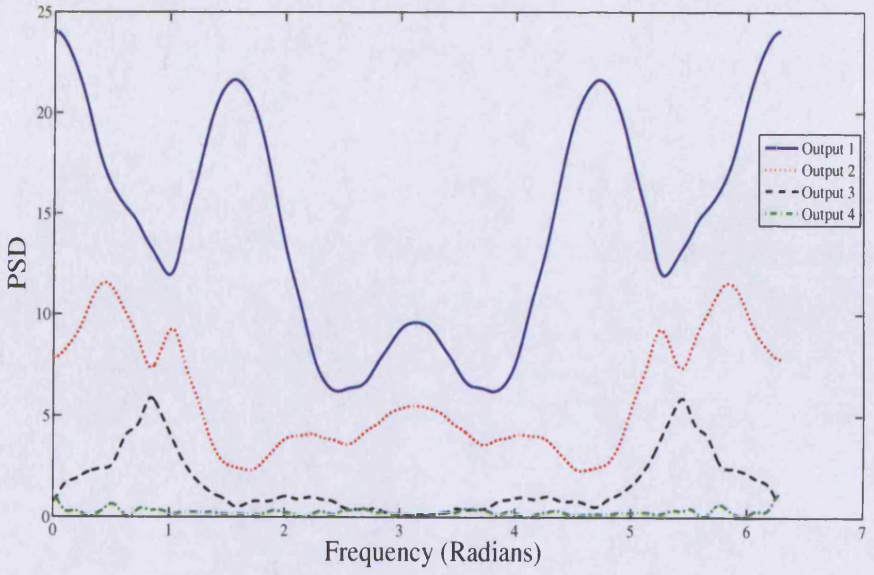

Figure 3.6: Plot of the spectra of the decorrelated signals, after applying the similarity transformation matrix $\underline{\mathbf{H}}(z)$ to the convolved signal $\mathbf{x}(t)$.

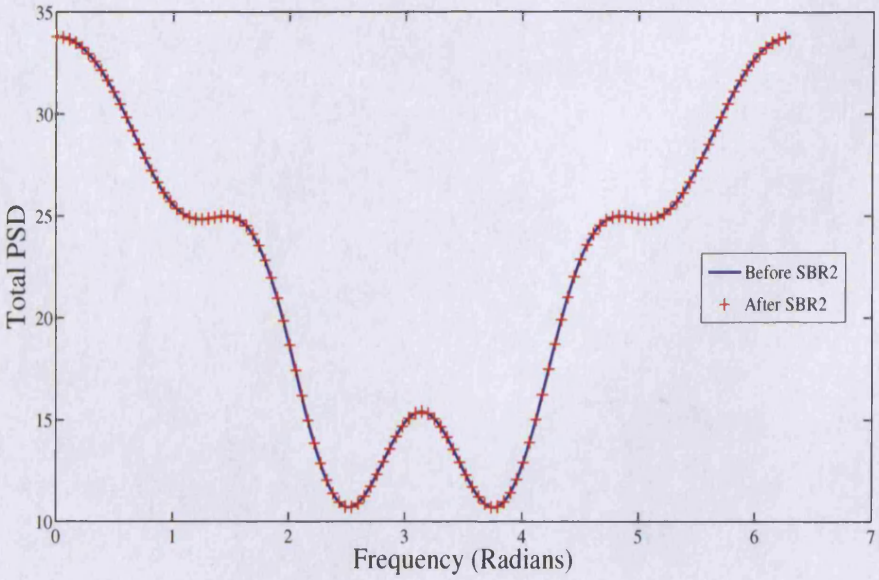

Figure 3.7: Plot of the total spectra of the expected signals, before and after applying the similarity transformation matrix $\underline{\mathbf{H}}(z)$ obtained from the SBR2 algorithm.

3.8 Uniqueness of Solutions

Note that for this decomposition the polynomial transformation matrix $\underline{\mathbf{H}}(z)$ is not unique. It is possible to have a diagonal paraunitary polynomial matrix $\underline{\mathbf{\Lambda}}(z) \in \underline{\mathbb{C}}^{p\times p}$, with diagonal elements consisting entirely of time-shift and phase adjustment terms, such that

$$\underline{\mathbf{\Lambda}}(z)\underline{\mathbf{H}}(z)\underline{\mathbf{R}}(z)\widetilde{\underline{\mathbf{H}}}(z)\widetilde{\underline{\mathbf{\Lambda}}}(z) = \underline{\mathbf{D}}(z) \tag{3.40}$$

where the diagonal elements of the matrix are of the form $\lambda_{jj}(t) = e^{i\alpha}z^{-t}$ for $j = 1, \ldots, p$ such that $\lambda_{jj}(t)\lambda^*_{jj}(-t) = \lambda^*_{jj}(-t)\lambda_{jj}(t) = 1$. However, as a result of the paraunitary condition, the diagonal matrix $\underline{\mathbf{D}}(z)$ will be unique since

$$\widetilde{\underline{\mathbf{\Lambda}}}(z)\underline{\mathbf{D}}(z)\underline{\mathbf{\Lambda}}(z) = \underline{\mathbf{D}}(z) \tag{3.41}$$

provided $\underline{\mathbf{D}}(z)$ is precisely diagonal. For the signal processing applications discussed in this

chapter, only the polynomial matrix $\underline{\mathbf{D}}(z)$ is generally of interest and so non-uniqueness of the paraunitary matrix $\underline{\mathbf{H}}(z)$ does not present a problem.

3.9 Limitations of the SBR2 Algorithm

At each iteration of the SBR2 algorithm, the order of the matrix being diagonalised can increase due to the application of the elementary delay in equation (3.11). Often after a series of iterations, the order of $\underline{\mathbf{R}}(z)$ becomes unnecessarily large. At each iteration new coefficient matrices are created at both ends of the polynomial matrix to accommodate the shifted coefficients, which now exceed the order of the initial polynomial matrix. These new coefficient matrices will consist entirely of zeros with the exception of the coefficients positioned in either the k^{th} row or column of the matrix. Over a series of iterations, further delays are applied to the matrix according to equation (3.11) and this can result in outer coefficient matrices containing mostly zero elements with the remainder accounting for only a small proportion of the Frobenius norm of the input matrix $\underline{\mathbf{R}}(z)$. Many of these coefficient matrices can be discarded without seriously compromising the accuracy of the decomposition. A similar problem is encountered with the paraunitary transformation matrix $\underline{\mathbf{H}}(z)$; the order of this matrix will also increase with the application of elementary delays, and can often become unnecessarily large.

With the orders of the polynomial matrices growing unnecessarily large, the computational load of the algorithm increases, resulting in a computationally slow algorithm, even when the initial input matrix is of a small size and order. The following chapter addresses this problem by proposing two methods of truncating the polynomial matrices within the algorithm, allowing an increased computational speed whilst maintaining a chosen level of accuracy for the matrix decomposition.

3.10 Conclusions

The SBR2 algorithm effectively transforms a para-Hermitian polynomial matrix to an approximately diagonal matrix by means of a series of paraunitary transformations. This is confirmed by the simple numerical example in Section 3.7. However in this example, the orders of the polynomial matrices $\underline{\widehat{\mathbf{D}}}(z)$ and $\underline{\mathbf{H}}(z)$ are seen to become unnecessarily large with many of the coefficients positioned in the outer lags of both matrices being either equal to zero or are very small in comparison to those coefficients in the central lags. This makes the algorithm unnecessarily slow to implement. The next chapter examines methods for preventing the matrices growing unnecessarily large, and so decreasing the computational time required for the SBR2 algorithm to run.

Chapter 4

Polynomial Matrix Truncation Methods

4.1 Introduction

To recap from Chapter 3, at the end of the i^{th} iteration of the SBR2 algorithm, the decomposition performed is of the form

$$\underline{\mathbf{H}}_i(z)\underline{\mathbf{R}}(z)\underline{\widetilde{\mathbf{H}}}_i(z) = \underline{\mathbf{D}}_i(z) \tag{4.1}$$

where $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times p}$ denotes the para-Hermitian polynomial input matrix to the algorithm, $\underline{\mathbf{H}}_i(z) \in \underline{\mathbb{C}}^{p\times p}$ is the paraunitary transformation matrix following i iterations and $\underline{\mathbf{D}}_i(z) \in \underline{\mathbb{C}}^{p\times p}$ is the resulting transformed polynomial matrix, which will be approximately diagonal, provided a sufficient number of iterations of the algorithm has been completed. The ability of the SBR2 algorithm to strongly decorrelate a set of convolutively mixed signals was demonstrated in the previous chapter by a simple numerical example. However, this example also highlighted a limitation of the algorithm; the unnecessarily large orders of the polynomial matrices $\underline{\mathbf{H}}_i(z)$ and $\underline{\mathbf{D}}_i(z)$, which grow with each iteration of the algorithm.

This point is clearly illustrated in Example 3.7, where the SBR2 algorithm was applied to a simple para-Hermitian polynomial matrix of a relatively small size and order, with only a few off-diagonal coefficients to drive to zero. However, even when applying the algorithm to this polynomial matrix, the order of the resulting diagonal matrix $\underline{\mathbf{D}}_{182}(z)$ grew to 3486,

which, by inspection of Figure 3.3, is seen to be unnecessarily large with the majority of the non-zero coefficients positioned in the central few lags of the polynomial matrix. The same is true of the transformation matrix $\underline{\mathbf{H}}_{182}(z)$ illustrated in Figure 3.4, whose order following the application of SBR2 was found to be 1733. These large and increasing orders of both polynomial matrices $\underline{\mathbf{D}}_i(z)$ and $\underline{\mathbf{H}}_i(z)$, made the SBR2 algorithm unnecessarily slow to implement due to the excessive computational load. However, they are also clearly unnecessary to obtain a sufficiently accurate polynomial matrix decomposition for most realistic purposes. Furthermore, for the potential application of the decomposition to MIMO communications[1], where the SBR2 algorithm is used to transform a MIMO channel into a set of Single-Input Single-Output (SISO) channels which are then equalised to obtain estimates of a set of transmitted signals, the order of the computed diagonal polynomial matrix is of critical importance. The computational complexity of the equaliser will at best be proportional to the length of the channel. Indeed, if a Maximum Likelihood Sequence Estimator (MLSE) based on the Viterbi algorithm is to be applied, as for the results used to illustrate this potential application in Chapter 8, the complexity grows exponentially. Clearly, the SBR2 algorithm is not practical for this application without using some technique to reduce the orders of the polynomial matrices.

4.1.1 The Problem

At each iteration of the SBR2 algorithm, the order of the polynomial matrices $\underline{\mathbf{H}}_i(z)$ and $\underline{\mathbf{D}}_i(z)$ of equation (4.1) increase, due to the application of elementary delay matrices. This point is easily illustrated by the following simple example. For ease of notation in this example, the $(j,k)^{th}$ polynomial elements of the matrix $\underline{\mathbf{D}}_i(z)$ are referred to as $\underline{d}^i_{jk}(z)$.

Suppose at the start of the i^{th} iteration of the algorithm, the dominant coefficient of the polynomial matrix $\underline{\mathbf{D}}_{i-1}(z)$ is found to be $d^{(i-1)}_{jk}(t)$. The coefficient $d^{(i-1)}_{kj}(-t)$ will then also be dominant due to the para-Hermitian structure of the polynomial matrix. As explained previously in Chapter 3, the objective of the i^{th} iteration of the SBR2 algorithm is to drive these two dominant coefficients to zero by firstly shifting the two coefficients so that they are

[1] A detailed description of this application can be found in Chapter 8.

positioned as the coefficient of z^0, by means of elementary delay matrices, and then apply the appropriate rotation matrix, which will force these coefficients to zero. The application of the elementary delay matrices will cause the order of the matrix $\underline{\mathbf{D}}_{i-1}(z)$ to increase by $2|t|$ to accommodate all of the shifted coefficients.

This point can be illustrated by examining the series of coefficients for the polynomial elements $\underline{d}_{jk}^{(i-1)}(z)$ and $\underline{d}_{kj}^{(i-1)}(z)$ before and after the application of the elementary delay matrices. For a simple example, suppose the polynomial matrix $\underline{\mathbf{D}}_{(i-1)}(z)$ is of order four and the dominant coefficients at iteration i are found to be $d_{jk}^{(i-1)}(-2)$ and $d_{kj}^{(i-1)}(2)$. Figure 4.1 demonstrates the magnitude of the series of coefficients associated with the two polynomial elements before and after the application of the elementary delay matrix. Notice that the order of each polynomial element has grown by four to accommodate all of the shifted coefficients. New coefficient matrices must be created at either end of the array to accommodate the shifted coefficients that now exceed the order of the polynomial matrix $\underline{\mathbf{D}}_{i-1}(z)$. These new coefficient matrices will consist entirely of zeros except for the shifted coefficients positioned in either the k^{th} row or column of the matrix. Note also that the elementary delay matrices are incorporated in the paraunitary transformation matrix and so each application of the elementary delay matrices, which are required to zero the dominant coefficients $d_{jk}^{(i-1)}(t)$ and $d_{kj}^{(i-1)}(-t)$, will also force the order of the polynomial paraunitary matrix $\underline{\mathbf{H}}_i(z)$ to increase by $|t|$.

During the subsequent step of the iteration, the elementary rotation matrix is applied to this matrix to drive the two dominant coefficients to zero. This rotation step will not alter the order of the matrix, but it will have an affect upon all polynomial coefficients in the j^{th} and k^{th} rows and columns of the matrix. The newly created coefficients from the application of the elementary delay matrix, which were equal zero, will have been affected by the rotation and will generally have increased in magnitude squared. Note that the Frobenius norm of each coefficient matrix $\mathbf{D}_i(t)$, $\forall t \in \mathbb{Z}$, is invariant to the application of the elementary rotation, but it will redistribute the Frobenius norm of the coefficients within each coefficient matrix.

Typically, after a series of such iterations, the repeated application of elementary delay

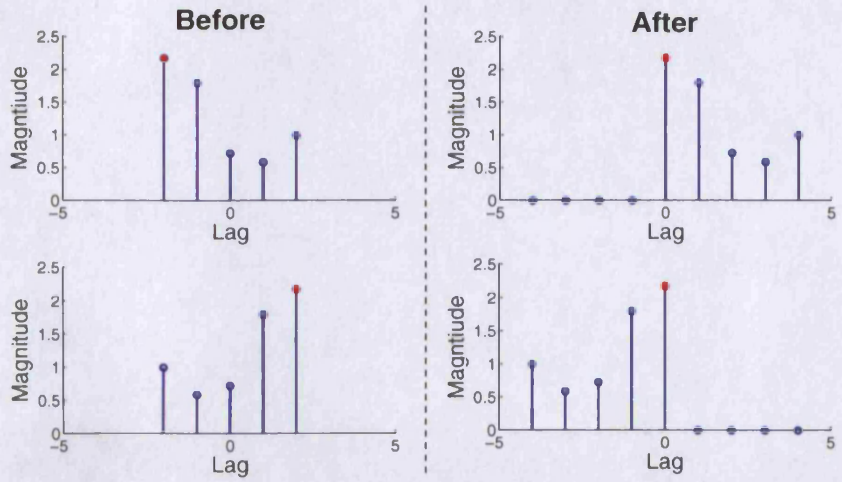

Figure 4.1: Stem plots of the series of coefficients of the polynomial elements $\underline{d}_{jk}^{(i-1)}(z)$ (top) and $\underline{d}_{kj}^{(i-1)}(z)$ (bottom) before (on the left) and after (on the right) the application of the elementary delay matrices. The dominant coefficient in each of the polynomial elements, before and after the appliacation of the delay matrix, is marked with a red dot.

matrices can force the order of both polynomial matrices within the algorithm to become unnecessarily large, with many of the polynomial coefficients positioned in the outer lags of the matrix equal to zero or, at most accounting for a small proportion of the Frobenius norm of the entire matrix. Clearly, these outer lags, which often contain little information, are not necessary to obtain an accurate polynomial matrix decomposition. However, storing them will typically make the algorithm computationally very slow to implement due to the increased computational load, and generally this will happen even if the input matrix $\underline{\mathbf{R}}(z)$ is of a relatively small size and order.

This chapter introduces two possible truncation methods which can be incorporated within the SBR2 algorithm to ensure that the order of the polynomial matrices do not grow unnecessarily large, thus reducing the computational load of the algorithm and decreasing the computational time taken for the algorithm to run. In the second part of this chapter, the two proposed truncation techniques are assessed by applying the SBR2 algorithm using each method in turn, to a series of para-Hermitian polynomial matrices of varying size and orders. In particular, the quality of the polynomial matrix decomposition when applying

each of the truncation techniques is examined, since truncating the order of the polynomial matrices will have some effect on the accuracy of the decomposition performed and it is important to ensure that the methods are used appropriately. A substantial amount of error in the decomposition would clearly have a detrimental effect on the capability of the algorithm for most applications. The chapter concludes with an overall recommendation as to which is the most appropriate truncation method to use and how it should be implemented as part of the SBR2 algorithm to effectively reduce the computational load of the algorithm, whilst maintaining an accurate polynomial matrix decomposition.

4.2 Truncation Method 1: Fixed Bound

The first method for truncating the polynomial matrices within the SBR2 algorithm is to place a fixed bound on the order of the polynomial matrices to ensure that they do not exceed this specified fixed limit. The truncation method is introduced for two cases, firstly a method suitable for a para-Hermitian polynomial matrix, such as the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$, and then a more general technique that can be applied to any polynomial matrix, such as the paraunitary transformation matrix $\underline{\mathbf{H}}_i(z)$. In either case, the polynomial matrix to be truncated will be denotes as $\underline{\mathbf{A}}(z)$.

4.2.1 For Para-Hermitian Polynomial Matrices

Suppose the polynomial matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{p\times p\times T}$ is para-Hermitian and therefore has coefficient matrices $\mathbf{A}(t)$ for $t = -T/2, \ldots, T/2$. To apply a fixed bound to this polynomial matrix, a limit $L > 0$ must be chosen and a suitable choice for this parameter must be an even number to ensure that the truncated polynomial matrix remains para-Hermitian. Any coefficient matrices $\mathbf{A}(t)$ for $|t| > L/2$ are truncated from the matrix and so the resulting truncated polynomial matrix can be expressed as

$$\underline{\mathbf{A}}^{tr}(z) = \sum_{t=-L/2}^{L/2} \mathbf{A}(t) z^{-t}. \tag{4.2}$$

Note that, as the polynomial matrix $\underline{\mathbf{A}}(z)$ is para-Hermitian, then the same number of coefficient matrices must be removed from both ends of the matrix to ensure that the matrix remains para-Hermitian. If the matrix is not para-Hermitian then a fixed bound could still be implemented, but there is now the option of setting both an upper and lower limit upon the lag variable t; this will now be discussed.

4.2.2 For Non-Para-Hermitian Polynomial Matrices

For a non-para-Hermitian polynomial matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{p\times q\times T}$, with coefficient matrices $\mathbf{A}(t)$ where $t = t_1, \ldots, t_2$, the order of the matrix can be calculated as $T = (t_2 - t_1)$. This is now not necessarily an even number and the coefficient matrices are not generally centred about the coefficient matrix associated with z^0. It is therefore far more difficult to apply a fixed bound truncation method to a non para-Hermitian polynomial matrix, as there is now the additional difficulty of determining in advance how many coefficient matrices should be truncated from either end of the polynomial matrix.

If this method is to be applied to the paraunitary polynomial matrix obtained within the SBR2 algorithm $\underline{\mathbf{H}}_i(z)$, then by inspection of Figure 3.4 it would appear that this matrix is also centred about the zero-lag coefficient matrix and so the same number of coefficient matrices can be removed from either end of the matrix. This is not guaranteed, but is generally found to be the case. If the fixed bound, for this instance, is L, then set $m = \lfloor \frac{T-L}{2} \rfloor$ and the truncated polynomial matrix can be calculated as

$$\underline{\mathbf{A}}^{tr}(z) = \sum_{t=t_1+m}^{t_2-m} \mathbf{A}(t) z^{-t} \tag{4.3}$$

where the order of this matrix will be either L or $L+1$, depending on whether the initial order of $\underline{\mathbf{A}}(z)$, prior to truncating, and the value L are even or odd.

Alternatively, if it is not suitable to remove the same number of coefficient matrices from both ends, a lower limit L_1 and an upper limit L_2 can be chosen, so that any coefficient matrices $\mathbf{A}(t)$ for $t > L_2$ and $t < L_1$ are truncated from the polynomial matrix, resulting in

the truncated polynomial matrix

$$\underline{\mathbf{A}}^{tr}(z) = \sum_{t=L_1}^{L_2} \mathbf{A}(t) z^{-t}, \tag{4.4}$$

which is of the fixed order $(L_2 - L_1)$. It is more appropriate to impose a lower and upper bound on the order of a general polynomial matrix as demonstrated by equation (4.4). However, it is difficult to determine the choice of the bounds in advance as a suitable choice will be related to the distribution of the Frobenius norm of the polynomial matrix over the series of lags. Ideally, the two separate bounds should be chosen from inspection of this measure, which is possible if a matrix is to be truncated just once. However, if this technique is to be implemented as part of an iterative routine, such as the SBR2 algorithm, where the order of the polynomial matrices can grow, it is not so appropriate as it will be impossible to determine appropriate values for these measures in advance.

4.3 Truncation Method 2: Energy Based Bound

The second method for restricting the growing orders of the polynomial matrices is based entirely upon the proportion of the Frobenius norm of the polynomial matrix being truncated which is permitted to be lost. Again, as with the previous truncation method, this technique is introduced for two cases, firstly a method suitable for para-Hermitian polynomial matrices and then a more general technique that can be applied to any polynomial matrix.

4.3.1 For Para-Hermitian Polynomial Matrices

For a polynomial para-Hermitian matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{p \times p \times T}$, with coefficient matrices $\mathbf{A}(t)$ for $t = -T/2, \ldots, T/2$, the truncation method finds the smallest value for the lag parameter $t_{lim} > 0$ such that

$$\frac{2 \sum_{t=t_{lim}}^{T/2} \sum_{j=1}^{p} \sum_{k=1}^{p} |a_{jk}(t)|^2}{\|\underline{\mathbf{A}}(z)\|_F^2} \leq \mu \tag{4.5}$$

where $a_{jk}(t)$ denotes the $(j,k)^{th}$ element of the coefficient matrix $\mathbf{A}(t)$ and μ defines the

proportion of the Frobenius norm of the untruncated polynomial matrix which is permitted to be lost due to the truncation. Once a value for t_{lim} has been found, the truncated matrix, $\underline{\mathbf{A}}^{tr}(z)$, can be calculated as

$$\underline{\mathbf{A}}^{tr}(z) = \sum_{-t_{lim}+1}^{t_{lim}-1} \mathbf{A}(t) z^{-t}. \tag{4.6}$$

There are two ways in which this technique can be used to truncate the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$ resulting from the i^{th} iteration of the SBR2 algorithm. Firstly, it can be permitted to allow a proportion of the Frobenius norm of the original input matrix $\underline{\mathbf{R}}(z)$ to be lost at each iteration. Or, alternatively, to allow a proportion of the Frobenius norm of $\underline{\mathbf{D}}_i(z)$, the matrix being truncated, to be lost at the end of the i^{th} iteration. The first method was adopted for the results presented in this chapter and so the denominator of equation (4.5) is set equal to $\|\underline{\mathbf{R}}(z)\|_F^2$ and $a_{jk}(t)$ denotes the $(j,k)^{th}$ element of $\mathbf{D}_i(t)$.

An additional constraint can be placed on the truncated polynomial matrix $\underline{\mathbf{D}}_i^{tr}(z)$ when applying this truncation method within the SBR2 algorithm to ensure that at most, only an acceptably small proportion of the Frobenius norm of $\underline{\mathbf{R}}(z)$ is lost over all iterations of the SBR2 algorithm. This type of constraint is implemented by requiring

$$\left\|\underline{\mathbf{D}}_i^{tr}(z)\right\|_F^2 \geq (1-\alpha) \left\|\underline{\mathbf{R}}(z)\right\|_F^2 \tag{4.7}$$

where α is the total proportion of $\|\underline{\mathbf{R}}(z)\|_F^2$ permitted to be lost over all iterations of the algorithm. This additional constraint can be used if it is ever essential to limit the proportion of the Frobenius norm of $\underline{\mathbf{R}}(z)$ which can be truncated over all iterations of the SBR2 algorithm. Although this type of constraint can be useful, it will not be used to obtain any of the results presented in this thesis.

4.3.2 For Non-Para-Hermitian Polynomial Matrices

A suitable truncation method for a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$, which is not necessarily para-Hermitian and has coefficient matrices $\mathbf{A}(t)$ for $t = T_1, \ldots, T_2$ can be implemented as

follows. Find a maximum value for t_1 and a minimum value for t_2 such that

$$\frac{\sum_{\tau=T_1}^{t_1}\sum_{l=1}^{p}\sum_{m=1}^{q}|a_{lm}(\tau)|^2}{\|\underline{\mathbf{A}}(z)\|_F^2} \leq \frac{\mu}{2} \tag{4.8}$$

and

$$\frac{\sum_{\tau=t_2}^{T_2}\sum_{l=1}^{p}\sum_{m=1}^{q}|a_{lm}(\tau)|^2}{\|\underline{\mathbf{A}}(z)\|_F^2} \leq \frac{\mu}{2} \tag{4.9}$$

where again μ defines the proportion of energy permitted to be truncated from the polynomial matrix $\underline{\mathbf{A}}(z)$ with one implementation of the truncation method. Then the coefficient matrices $\mathbf{A}(\tau)$ for $\tau = T_1, \ldots, t_1$ and $\tau = t_2, \ldots, T_2$ can be truncated from the matrix and the truncated polynomial matrix calculated as

$$\underline{\mathbf{A}}^{tr}(z) = \sum_{t_1+1}^{t_2-1} \mathbf{A}(t)z^{-t}. \tag{4.10}$$

This truncation method can be applied to the paraunitary polynomial matrix obtained at the end of the i^{th} iteration of the SBR2 algorithm, $\underline{\mathbf{H}}_i(z)$. If the polynomial matrix $\underline{\mathbf{A}}(z)$ in equations (4.8) and (4.9) is para-Hermitian then this truncation method simply reduces to the one demonstrated by equation (4.5).

4.4 Comparing the Truncation Methods

To assess the different truncation methods and further illustrate the SBR2 algorithm, a set of polynomial para-Hermitian test matrices was generated.

4.4.1 Set of Test Matrices

The first polynomial matrix of the set $\underline{\mathbf{R}}_1(z) \in \mathbb{C}^{3\times3\times4}$ was generated by randomly drawing both the real and imaginary parts of the complex coefficients for each of the polynomial elements from a Gaussian distribution with mean zero and unit variance. Care must be taken to ensure that the polynomial matrix is para-Hermitian by ensuring the coefficients

associated with the polynomial elements satisfy

$$r_{jk}(t) = r_{kj}^{*}(-t) \tag{4.11}$$

$\forall t \in \mathbb{Z}$ and for $j, k = 1, 2, 3$.

The second polynomial para-Hermitian matrix $\underline{\mathbf{R}}_2(z) \in \mathbb{R}^{4\times4\times4}$ was chosen to be a fairly sparse polynomial matrix, where the non-zero coefficients associated with each of the polynomial elements were drawn randomly from a uniform distribution in the range $[0, 1]$. This matrix has a total of 38 non-zero coefficients, all of which are real.

For a more practically motivated example $\underline{\mathbf{R}}_3(z) \in \mathbb{R}^{5\times5\times30}$ is the estimated space-time covariance matrix for a set of five convolved signals, where, without loss of generality, the mixing matrix for the system was chosen to demonstrate the propagation of three signals onto five sensors. The source signals were chosen to be independent binary phase-shift keying (BPSK) sequences, which means each sample can take the value ±1 with equal probability of a half. These signals were mixed according to the mixing model demonstrated in Section 3.6.1, where the coefficients associated with the polynomial elements of the channel matrix $\underline{\mathbf{C}}(z) \in \mathbb{R}^{5\times3\times4}$, required for equation (3.26), were drawn from a uniform distribution in the range $[-1, 1]$. The variance of the additive noise was chosen to be unity, which meant an SNR value of 5.4dB. This was specifically chosen, as a point where the SBR2 algorithm can effectively separate the signal and noise subspaces. The estimated space-time covariance matrix was calculated according to equation (3.30), where the correlation lag window was set as $W = 15$.

The fourth polynomial matrix $\underline{\mathbf{R}}_4(z) \in \mathbb{R}^{5\times5\times10}$ was chosen to be the expected or ideal space-time covariance matrix corresponding to the signals used to calculate the estimated space-time covariances matrices $\underline{\mathbf{R}}_3(z)$. This matrix was therefore calculated as

$$\underline{\mathbf{R}}_4(z) = \underline{\mathbf{C}}(z) \underbrace{\underline{\mathbf{R}}_{ss}(z)}_{=\mathbf{I}_3} \widetilde{\underline{\mathbf{C}}}(z) + \sigma^2\mathbf{I}_5 \tag{4.12}$$

where $\underline{\mathbf{R}}_{ss}(z)$ denotes the space-time covariance for the source signals, which for the example presented here, is given by the identity matrix, $\underline{\mathbf{C}}(z)$ denotes the appropriate channel matrix

and σ^2 denotes the variance of the noise process $\mathbf{n}(t)$.

Similarly, the para-Hermitian polynomial matrix $\underline{\mathbf{R}}_5(z) \in \mathbb{C}^{5\times5\times20}$ is also an estimated space-time covariance matrix for a set of convolved signals. The coefficients associated with the polynomial elements of the mixing matrix $\underline{\mathbf{C}}(z) \in \mathbb{C}^{5\times3\times4}$ are complex, with both the real and imaginary parts drawn from a uniform distribution in the range $[-1, 1]$. The three source signals in this case were chosen to be independent quaternary phase-shift keying (QPSK) sequences of length 1000. The correlation window length for this matrix was set as $W = 10$. It would have been sufficient for both examples to set W equal to the order of the polynomial mixing matrix, due to the statistics of the data, but in most instances, no knowledge about the mixing is known and so this parameter must be estimated.

The final test matrix $\underline{\mathbf{R}}_6(z) \in \mathbb{C}^{5\times5\times10}$ was chosen to be the expected space-time covariance matrix for the signals used to calculate the estimated space-time covariances matrix $\underline{\mathbf{R}}_5(z)$. This matrix was therefore calculated in a similar way to the polynomial matrix $\underline{\mathbf{R}}_4(z)$ demonstrated in equation (4.12). A summary of the properties of each of the polynomial para-Hermitian test matrices can be found in Table 4.1 and will now be discussed.

Matrix	Order	Number of Non-Zero Off-Diagonal Elements	$\|\|\underline{\mathbf{R}}(z)\|\|_F^2$	Proportion of $\|\|\underline{\mathbf{R}}(z)\|\|_F^2$ Positioned in the Off-Diagonal Elements
$\underline{\mathbf{R}}_1(z)$	4	30	66.05	0.8115
$\underline{\mathbf{R}}_2(z)$	4	38	29.72	0.8244
$\underline{\mathbf{R}}_3(z)$	30	620	516.36	0.4972
$\underline{\mathbf{R}}_4(z)$	10	220	455.96	0.4658
$\underline{\mathbf{R}}_5(z)$	20	420	2020.31	0.4781
$\underline{\mathbf{R}}_6(z)$	10	220	2054.77	0.4362

Table 4.1: Properties of each of the para-Hermitian polynomial test matrices.

4.4.2 Comments on the Polynomial Test Matrices

Firstly, before testing the truncation methods, the SBR2 algorithm was applied to each of the test matrices, using the same type of stopping condition as the numerical example in Chapter

3, where the algorithm was set to stop when the magnitude of the dominant coefficient, g, was deemed sufficiently small compared to the diagonal zero-lag coefficients. For the test matrices defined in Section 4.4.1, the SBR2 algorithm was stopped once

$$g < 10^{-2}\sqrt{N_1/p}, \tag{4.13}$$

where the measure N_1 has been defined in equation (3.20) and p defines the number of rows of the para-Hermitian test matrix.

Table 4.2 contains the number of iterations required for each of the polynomial test matrices to converge according to the stopping condition demonstrated by equation (4.13). Upon inspection of this and the results contained in Table 4.1, it is evident that there is a relationship between the number of non-zero off-diagonal coefficients and the number of iterations required for the SBR2 algorithm to converge according to equation (4.13). Clearly, the more off-diagonal non-zero coefficients, the more iterations required to diagonalise the matrix. However, there appears to be no relationship between the initial order of the polynomial matrix or the number of iterations required to converge and the final order of the diagonal matrix following the decomposition.

Matrix	Number of Iterations	g	Order of $\underline{\mathbf{D}}(z)$
$\underline{\mathbf{R}}_1(z)$	91	0.0449	1088
$\underline{\mathbf{R}}_2(z)$	123	0.0268	1294
$\underline{\mathbf{R}}_3(z)$	271	0.0975	6242
$\underline{\mathbf{R}}_4(z)$	198	0.0939	2394
$\underline{\mathbf{R}}_5(z)$	330	0.1915	6052
$\underline{\mathbf{R}}_6(z)$	351	0.1958	6666

Table 4.2: For each of the test matrices, the number of iterations required to satisfy the stopping condition expressed in equation (4.13), the resulting order of the approximately diagonal polynomial matrix and the final value of the dominant coefficient, g, once the stopping condition has been satisfied.

4.5 Truncation Method Results

This section demonstrates how the two truncation methods can affect the performance of the SBR2 algorithm and, in particular, the accuracy of the resulting polynomial matrix decomposition. The SBR2 algorithm was applied to each of the polynomial para-Hermitian test matrices detailed in Section 4.4.1, for the following three cases,

1. when no truncation method is used,
2. using Truncation Method 1 (the fixed bound) and
3. using Truncation Method 2 (the energy based bound).

For each case, no stopping condition was used within the SBR2 algorithm, allowing each implementation of the algorithm to complete 200 iterations. For the cases where a truncation method is used, it was applied at the end of each iteration i to truncate the resulting transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$ and this was carried out for varying levels of truncation. The paraunitary transformation matrix $\underline{\mathbf{H}}_i(z)$ can also be truncated using either of the truncation methods suitable for non-para-Hermitian polynomial matrices. However, this is not done for the results presented here as this matrix is not calculated as part of the main iterative process of the SBR2 algorithm and we want to make enable a fair comparison between the truncation methods. Note that implementing either of the two truncation methods within the SBR2 algorithm will not affect the proof of convergence detailed in Section 3.4, as the quantity N_1 will still increase monotonically and it remains bounded from above by the initial value of N_4. Therefore, when using either of the truncation methods, the polynomial matrix $\underline{\mathbf{D}}_i(z)$ is guaranteed to converge to a diagonal polynomial matrix.

4.5.1 Accuracy of the Decomposition

To assess the quality of the decomposition performed by the SBR2 algorithm, whilst implementing one of the two truncation methods, the relative error between the input matrix, $\underline{\mathbf{R}}(z)$, and the matrix obtained from the inverse decomposition, $\underline{\mathbf{R}}'(z) = \underline{\widetilde{\mathbf{H}}}(z)\underline{\mathbf{D}}(z)\underline{\mathbf{H}}(z)$, was calculated, where $\underline{\mathbf{H}}(z)$ and $\underline{\mathbf{D}}(z)$ denote the paraunitary polynomial matrix and the approx-

imately diagonal polynomial matrix obtained using the SBR2 algorithm. The relative error[2] for the decomposition is therefore defined to be

$$E_{rel} = \frac{\left\| \underline{\mathbf{R}}(z) - \underline{\mathbf{R}}'(z) \right\|_F}{\| \underline{\mathbf{R}}(z) \|_F}. \tag{4.14}$$

4.5.2 Case 1: No Truncation

The SBR2 algorithm was applied in turn to each of the polynomial test matrices from Section 4.4.1. As demonstrated by Example 3.7 of the previous chapter, the order of each resulting diagonal polynomial matrix was found to be unnecessarily large. Furthermore, for each case, the majority of the Frobenius norm of the resulting approximately diagonal polynomial matrix is positioned in the couple of hundred lags centred about the zero lag coefficient matrix, with higher order terms negligibly small or equal to zero. This is clearly illustrated by Figure 4.2, which demonstrates the measure

$$C(t) = \frac{\| \mathbf{D}_{200}^K(0) \|_F^2 + 2 \sum_{\tau=1}^{t} \| \mathbf{D}_{200}^K(\tau) \|_F^2}{\| \underline{\mathbf{R}}_K(z) \|_F^2} \tag{4.15}$$

over the lags $t = 1, \ldots, 201$, where $\mathbf{D}_{200}^K(\tau)$ denotes the coefficient matrix of $z^{-\tau}$ in the transformed polynomial matrix obtained from the SBR2 algorithm when applied to each of the polynomial test matrices $\underline{\mathbf{R}}_K(z)$ for $K = 1, \ldots, 6$. In fact, for each of the observed transformed matrices over 95% of the Frobenius norm of each matrix is positioned in the central three lags of the matrix and more than 99% in the central 21.

Figure 4.3 shows how the order of each of the six test matrices increases when the SBR2 algorithm is applied to each in turn for a series of 200 iterations using no truncation method. The order of the final diagonalised matrix following all iterations of the algorithm can be found in Table 4.4. Note that there appears to be no relationship between either the order of the para-Hermitian input matrix or the number of off-diagonal non-zero coefficients in the matrix,

[2] In this chapter, this is used to measure only the effect of the polynomial matrix truncation methods on the accuracy of the decomposition performed and does not account for the level of error encountered by only calculating an approximately diagonal polynomial matrix. For the application of the SBR2 algorithm, a more appropriate relative error can be defined to take into account both of these factors.

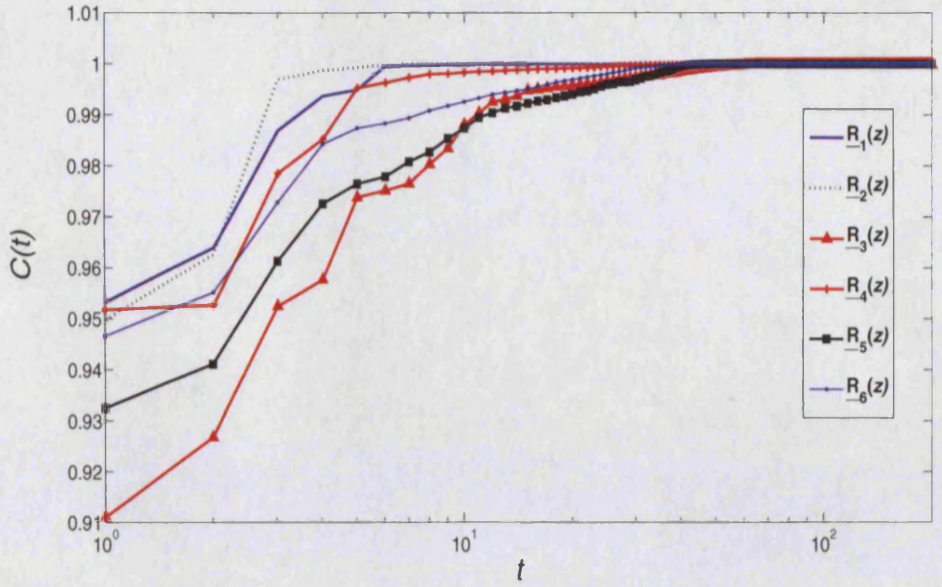

Figure 4.2: The cumulative squared Frobenius norm of the matrix $\underline{\mathbf{D}}_{200}(z)$ ($C(t)$) over all lags (t) when no truncation method is used, calculated from the centre outwards, for all test matrices.

both contained in Table 4.1, and the order of the resulting transformed polynomial matrix following 200 iterations of the SBR2 algorithm. Furthermore, the magnitude or distribution of the off-diagonal coefficients over the series of coefficient matrices is also of no relevance in predicting the order of the final matrix.

Conclusions: No Truncation Method

These results confirm the clear requirement for a polynomial matrix truncation method within the SBR2 algorithm. For each example, the order of the resulting diagonal polynomial matrix was found to be unnecessarily large, with many of the coefficients associated with outer lags of the matrix accounting for a very small proportion of the Frobenius norm of the matrix. This point is particularly important for the application of the decomposition to communication systems, where the SBR2 algorithm, or other algorithms to be derived later in this thesis, can be used to separate a MIMO communication channel into a set of independent subchannels, which are then equalised using an existing SISO equalisation technique. For this application

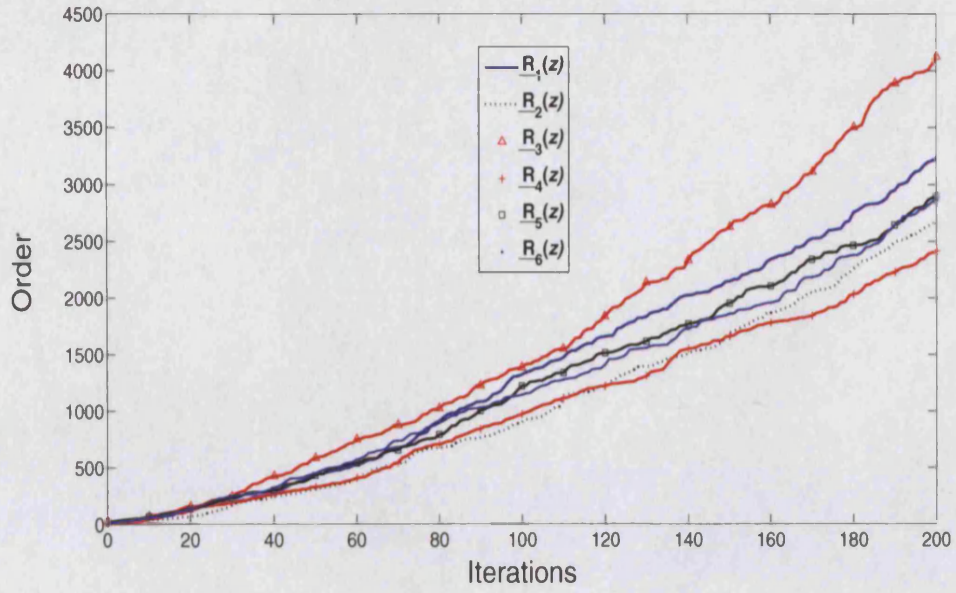

Figure 4.3: The order of the transformed polynomial matrix at each of the 200 iterations of the SBR2 algorithm when applied to each of the para-Hermitian test matrices using no truncation method.

of the algorithm, the order of the resulting diagonal matrix obtained by the algorithm must be sufficiently small to enable equalisation of the set of SISO channels. For example, if a maximum likelihood sequence esimator (MLSE) based on the Viterbi algorithm is to be implemented, then the computational complexity of the scheme is exponentially proportional to the order of this matrix.

Furthermore, without a truncation routine the SBR2 algorithm is unnecessarily slow to implement, with the order of both the paraunitary matrix and the transformed polynomial matrices $\underline{\mathbf{H}}_i(z)$ and $\underline{\mathbf{D}}_i(z)$ increasing at each iteration and often resulting in many of the outer coefficient matrices consisting entirely of zeros. Table 4.5 contains the computational time taken for the SBR2 algorithm to complete 200 iterations when applied to each of the polynomial test matrices using no truncation method.

4.5.3 Case 2: Fixed Bound Truncation Method

The SBR2 algorithm was again applied to each of the para-Hermitian polynomial test matrices detailed in Section 4.4.1, however, this time applying the fixed bound truncation method described in Section 4.2.1 to truncate the order of the polynomial matrix $\underline{\mathbf{D}}_i(z)$ from equation (4.1) at the end of each iteration i. The main problem with this truncation method is that it is difficult to know in advance what value to use for the fixed bound parameter L and to find an appropriate value for this parameter, which will not compromise the accuracy of the decomposition significantly, will involve a considerable amount of trial and error. If the value is too large, then the algorithm will be unnecessarily slow to implement, whilst if L is set too small then the Frobenius norm of the polynomial input matrix to the SBR2 algorithm $\underline{\mathbf{R}}(z)$ has been unnecessarily compromised, leading to inaccurate results.

From inspection of Figure 4.2, it can be seen that the majority of the polynomial coefficients of the resulting six approximately diagonal polynomial matrices following 200 iterations of the SBR2 algorithm are positioned in the couple of hundred coefficient matrices centred about the zero lag. Therefore, the fixed bound parameter L was chosen to take all even integers in the range $[1, 200]$. Figure 4.4 demonstrates the relative error of the decomposition E_{rel} obtained from applying the SBR2 algorithm to each of the test matrices for this range of values of L. From this figure a suitable choice for L can be made for each of the test matrices, depending on the required accuracy of the decomposition. A suitable choice of L to obtain a very accurate decomposition can be seen to be somewhere between 100 and 150 for all examples, which is considerably smaller than the order obtained if no truncation method is used. The exact value L required to obtain various values of E_{rel} can be seen in Table 4.3. The total computational time to implement 200 iterations of the SBR2 algorithm using this truncation method for the range of values for L and for each of the test matrices can be seen in Figure 4.5. Even when using a fixed bound of $L = 200$ the computational time has been vastly reduced from the time taken when no truncation method is used, as shown in Table 4.5. These results demonstrate that the computational time can be vastly reduced, whilst still calculating a reasonably accurate decomposition.

Finally, it might be expected that a suitable choice of L could be determined by either

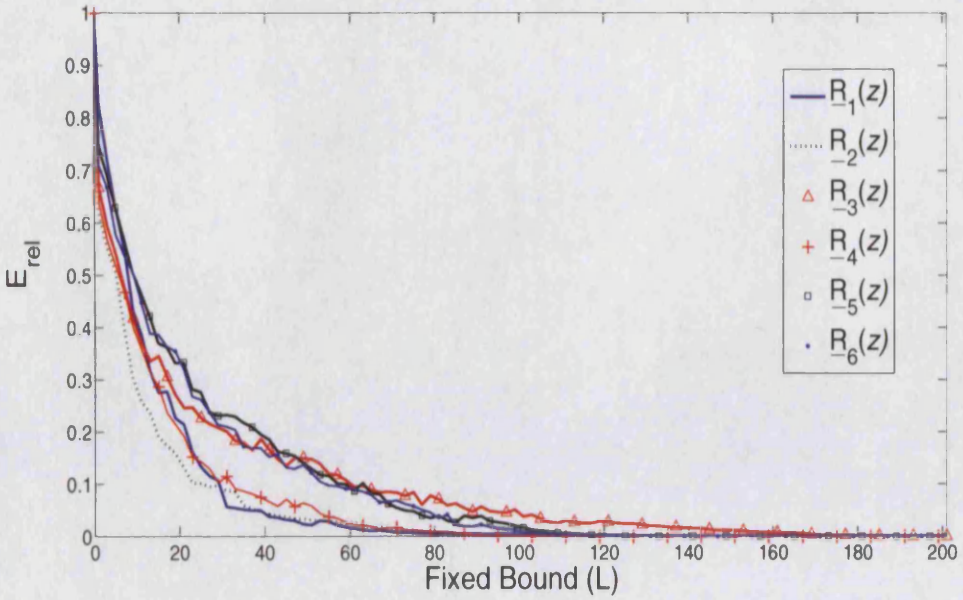

Figure 4.4: The relative error of the decomposition, E_{rel}, obtained for each value of the fixed bound parameter L, for each of the six test matrices.

Matrix	Minimum value of L to obtain $E_{rel} =$				
	10^{-4}	10^{-3}	10^{-2}	10^{-1}	2×10^{-1}
$\underline{\mathbf{R}}_1(z)$	146	132	62	24	18
$\underline{\mathbf{R}}_2(z)$	104	90	70	20	12
$\underline{\mathbf{R}}_3(z)$	210	180	144	50	20
$\underline{\mathbf{R}}_4(z)$	114	96	72	24	16
$\underline{\mathbf{R}}_5(z)$	152	126	102	48	26
$\underline{\mathbf{R}}_6(z)$	146	124	96	44	26

Table 4.3: The minimum value of the fixed bound parameter L to obtain a particular level of relative error for the polynomial matrix decomposition obtained by the SBR2 algorithm when using the fixed bound truncation method.

the number of non-zero off-diagonal polynomial coefficients, the number of coefficients whose magnitude is larger that the stopping criterion ϵ or the order of the input matrix to the algorithm. However, upon inspection of the results there appears to be no relationship present between any of these quantities. From Figure 4.4, the relative error for the two test matrices of order four, $\underline{\mathbf{R}}_1(z)$ and $\underline{\mathbf{R}}_2(z)$, appears to be quite similar for each of the fixed bound values. However, from this figure the two polynomial matrices $\underline{\mathbf{R}}_4(z)$ and $\underline{\mathbf{R}}_6(z)$, which are both of order 10 and also have the same number of off-diagonal elements, can be seen to have very different behavior to the various fixed bounds. Furthermore, polynomial matrices $\underline{\mathbf{R}}_3(z)$ and $\underline{\mathbf{R}}_4(z)$ are the estimated and expected space-time covariance matrices for the same set of convolutively mixed BPSK signals respectively. However, although these matrices are very similar in structure, with any deviations due to estimation errors, there appears to be no relationship between how the order of each of the polynomial test matrices increases over the series of iterations. The same is true of the polynomial matrices $\underline{\mathbf{R}}_5(z)$ and $\underline{\mathbf{R}}_6(z)$, which also have a similar relationship.

For the test matrices here, $L = 200$ would certainly give an accurate level of matrix decomposition and also drastically reduce the computational time. For the majority of the matrices, $L = 100$ would be more than suitable. Note that when using this truncation method, it is important to always calculate the relative error of the decomposition to ensure that the chosen fixed bound value L is large enough to ensure a sufficiently accurate decomposition.

Conclusions: Fixed Bound Truncation Method

The results have demonstrated that applying an appropriate limit on the order of the polynomial matrix $\underline{\mathbf{D}}_i(z)$ within the SBR2 algorithm can work well and, if used appropriately, can vastly reduce the computational time taken to implement the algorithm, whilst also ensuring that an accurate polynomial matrix decomposition is calculated. However, when using this truncation method it is difficult to predetermine the choice of the fixed bound parameter L suitable for a particular input matrix to optimise both the speed of the algorithm and accuracy of the decomposition performed. Furthermore, finding an appropriate value for each matrix will involve a process of trial and error. When using this truncation method, it is

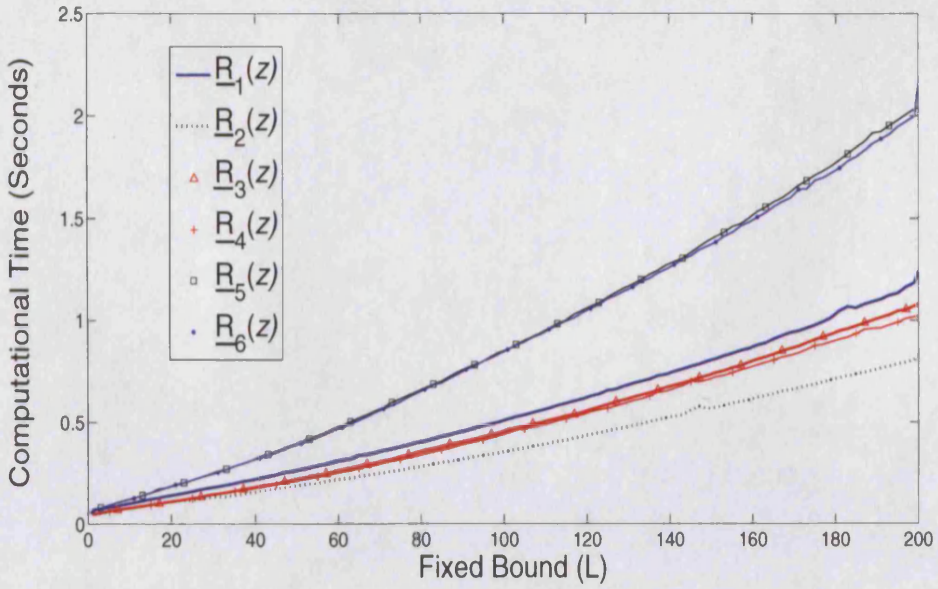

Figure 4.5: Computational time (in seconds) taken by the SBR2 algorithm when applied to each of the six test matrices for varying values of the fixed bound L.

important to always calculate the relative error of the decomposition performed, to ensure that it is sufficiently accurate for the required application. Other measures that are useful to calculate are the proportion of the Frobenius norm of the input matrix that is lost and also, if truncating the paraunitary matrix $\underline{\mathbf{H}}_i(z)$, the Frobenius norm of this matrix that is lost. It is important to look at both, to ensure that one of the matrices has not been unreasonably truncated. One advantage of using this method, is that it is possible to specify the dimensions of the polynomial matrices in advance, which can be advantageous in terms of the computational time to implement the algorithm.

4.5.4 Case 3: Energy Based Truncation Method

For the third and final case, the SBR2 algorithm was applied to the set of test matrices from Section 4.4.1, this time implementing the energy based truncation method detailed in Section 4.3.1. Again, only the transformed para-Hermitian polynomial matrix $\underline{\mathbf{D}}_i(z)$ from equation (4.1) was truncated, by applying the truncation method at the end of each iteration of the algorithm.

Firstly, setting the truncation parameter μ equal to zero[3] for each implementation of the SBR2 algorithm vastly reduced the order of the resulting polynomial matrix $\underline{\mathbf{D}}_{200}(z)$. This can be seen in Table 4.4, which contains the orders of this polynomial matrix obtained from calculating the decomposition of each of the polynomial test matrices when no truncation method is used and then for the cases $\mu = 0$ and $\mu = 10^{-8}$. It can be seen from this table that increasing the value of μ will further reduce the order of the final polynomial matrix $\underline{\mathbf{D}}_{200}(z)$ for each case.

Matrix	Order of $\underline{\mathbf{D}}_{200}(z)$		
	No Truncation Method	**Remove Zeros Only ($\mu = 0$)**	**Method 2 ($\mu = 10^{-8}$)**
$\underline{\mathbf{R}}_1(z)$	3238	1720	74
$\underline{\mathbf{R}}_2(z)$	2684	1316	68
$\underline{\mathbf{R}}_3(z)$	4130	1872	182
$\underline{\mathbf{R}}_4(z)$	2420	1210	110
$\underline{\mathbf{R}}_5(z)$	2898	1498	138
$\underline{\mathbf{R}}_6(z)$	2880	1334	114

Table 4.4: Order of the diagonalised polynomial matrix $\underline{\mathbf{D}}_i(z)$ obtained from applying the SBR2 algorithm to each of the test matrices, when using no truncation method, using the energy based truncation method with $\mu = 0$ and $\mu = 10^{-8}$.

These results demonstrate that when using the second polynomial matrix truncation method, it is possible to remove only coefficient matrices at the outer edges of the polynomial matrix that consist entirely of zeros (to computational precision) by setting $\mu = 0$. The resulting relative error of the inverse decomposition for this case will always be equal to zero. Figure 4.6 illustrates how the order of the transformed matrix will grow throughout the SBR2 algorithm, when applied to each of the polynomial test matrices, for the case $\mu = 0$. The order of the matrices can be seen to be vastly reduced for this choice of μ, when compared to the results where no truncation method is used, shown in Figure 4.3 and Table 4.4. The order is approximately halved for each of the test matrices. Accordingly, the computational

[3]This is numerical zero and not 10^{-16}.

time taken to run the SBR2 algorithm for each matrix is also greatly reduced, as confirmed in Table 4.5.

Furthermore, if the truncation parameter μ is set larger than zero, then both the order of the matrix and the time taken to implement the decomposition can be again reduced. This is illustrated by the results obtained for the case $\mu = 10^{-8}$, recorded in Tables 4.4 and 4.5. The relative error of the decomposition has also not been significantly compromised using this value of μ and for each of the test matrices is found to be less than 10^{-3}. Clearly, it is not necessary to store all of these outer coefficient matrices to still obtain an accurate polynomial matrix decomposition.

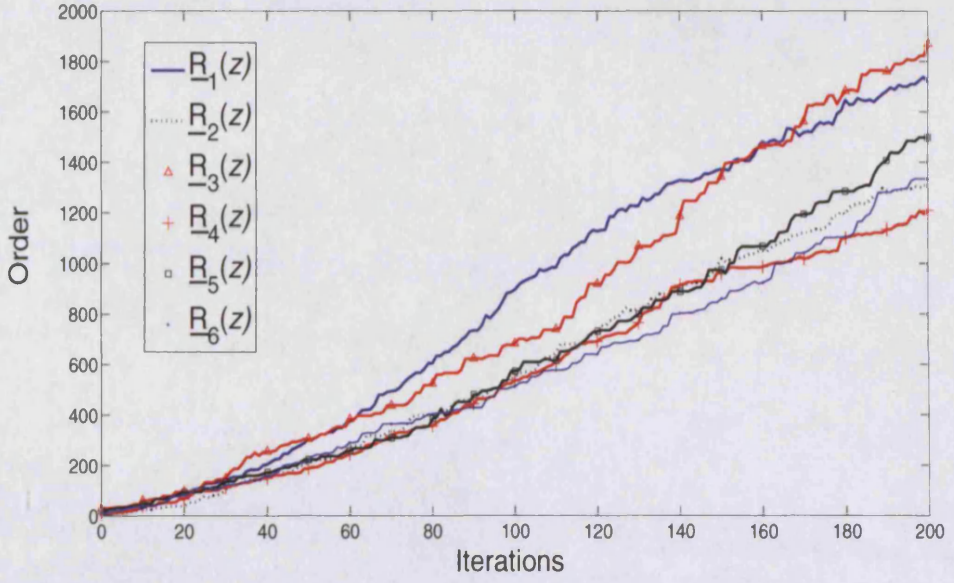

Figure 4.6: The order of each of the polynomial test matrices at each iteration of the algorithm when outer coefficient matrices consisting entirely of zeros are truncated.

Matrix	Computational Time (Seconds)		
	No Truncation Method	Remove Zeros Only ($\mu = 0$)	Method 2 ($\mu = 10^{-8}$)
$\underline{\mathbf{R}}_1(z)$	47.76	18.71	0.14
$\underline{\mathbf{R}}_2(z)$	20.82	7.22	0.13
$\underline{\mathbf{R}}_3(z)$	97.12	16.93	1.01
$\underline{\mathbf{R}}_4(z)$	25.80	8.41	0.32
$\underline{\mathbf{R}}_5(z)$	77.20	20.84	1.76
$\underline{\mathbf{R}}_6(z)$	73.07	17.26	1.97

Table 4.5: Computational time taken to apply the SBR2 algorithm to each of the polynomial para-Hermitian test matrices for the varying truncation methods.

Maximum Energy Loss

Applying the energy based truncation method at the end of each iteration of the SBR2 algorithm to truncate the polynomial matrix $\underline{\mathbf{D}}_i(z)$ to allow at most the proportion μ of the Frobenius norm of the initial polynomial matrix to be lost, the maximum amount of energy lost following N iterations of the algorithm can be calculated as

$$Loss\,(N) = \sum_{k=1}^{N} \mu\,(1-\mu)^{k-1}\,\|\underline{\mathbf{R}}(z)\|_F^2 \tag{4.16}$$

$$= \left[1-(1-\mu)^N\right]\|\underline{\mathbf{R}}(z)\|_F^2 \tag{4.17}$$

where $\mu \in [0,1]$, $\|\underline{\mathbf{R}}(z)\|_F^2$ defines the squared Frobenius norm of the input matrix to the algorithm $\underline{\mathbf{R}}(z)$ and is consistent with the notation used previously in Chapter 3. Therefore, the minimum possible amount of energy remaining in the resulting matrix following N iterations, $\underline{\mathbf{D}}_N(z)$, is

$$Rem\,(N) = (1-\mu)^N\,\|\underline{\mathbf{R}}(z)\|_F^2\,. \tag{4.18}$$

Figure 4.7 illustrates the minimum proportion of $\|\underline{\mathbf{R}}(z)\|_F^2$ remaining with the associated choice of the truncation parameter μ to guarantee this, if the algorithm is to complete a range of different number of iterations. For most implementations of the SBR2 algorithm

when truncating the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$, it is unlikely that the maximum amount of energy will be lost at each iteration. However, the quantity shown by equation (4.18) represents a lower bound on the proportion of $\|\underline{\mathbf{R}}(z)\|_F^2$ remaining for a particular value of μ over a specified number of iterations.

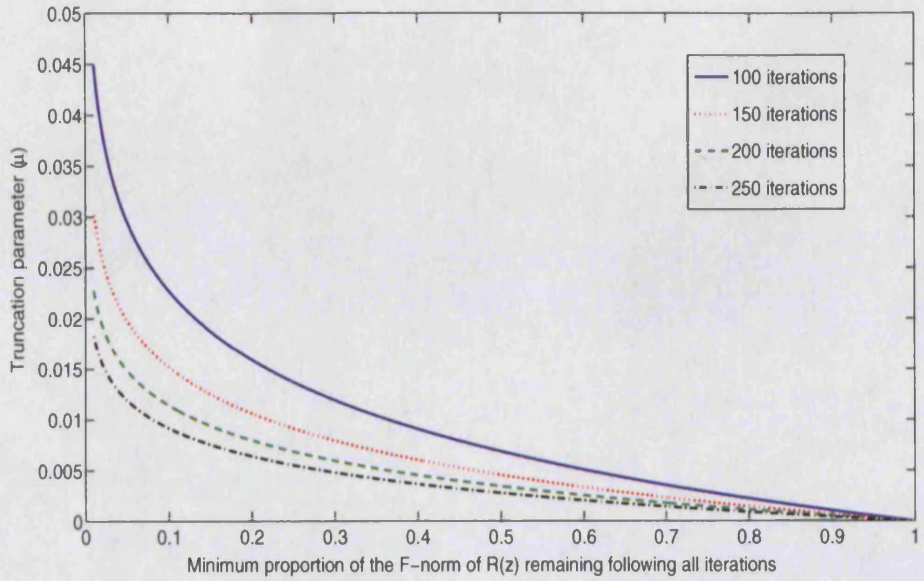

Figure 4.7: The minimum proportion of $\|\underline{\mathbf{R}}(z)\|_F^2$ remaining following truncating the polynomial matrix within the SBR2 algorithm, for various numbers of iterations.

If an estimate of the number of iterations that the algorithm will complete can be generated, then using equation (4.18), a value for the truncation parameter μ can be calculated, given the proportion of energy required to remain within the polynomial matrix. This measure can be calculated before applying the SBR2 algorithm to the polynomial matrix and can therefore be used in advance to help choose a value for the truncation parameter μ when using the energy based truncation method. However, this would require knowledge of how many iterations the algorithm is going to run for, and this cannot be determined precisely in advance. However, note that this measure is related to the number of non-zero off-diagonal coefficients contained in the para-Hermitian polynomial input matrix to the algorithm; this point has been mentioned previously in Section 4.4.1.

Conclusions: Energy Based Truncation Method

The energy based truncation method is clearly the most appropriate of the two truncation methods as it allows some control over how much of the Frobenius norm of the polynomial matrix $\underline{\mathbf{R}}(z)$ is lost at each iteration of the decomposition algorithm, and this determines the accuracy of the decomposition. The results presented above demonstrate that setting $\mu = 0$, which will only remove coefficient matrices positioned in the outer lags consisting entirely of zeros, can generally drastically reduce the order of the diagonal polynomial matrix obtained from the decomposition and, as a consequence, can reduce the computational time taken to implement the SBR2 algorithm. Typically, using this value of μ will approximately half the order of the diagonal polynomial matrix obtained from the decomposition and the computational time will be at least halved, although often it is reduced considerably more than this. If the truncation parameter is set greater than zero, then the order of the matrix can be further reduced, although care must be taken to ensure that the accuracy of the decomposition performed is not significantly compromised.

4.6 Numerical Example 3.7 With Truncation

In the numerical example from Chapter 3, the SBR2 algorithm was applied to the polynomial space-time covariance matrix for a set of convolutively mixed BPSK signals $\underline{\mathbf{R}}(z) \in \mathbb{C}^{4\times4\times20}$. This simple example demonstrated the ability of the SBR2 algorithm to calculate the EVD of this para-Hermitian polynomial matrix and consequently strongly decorrelate the set of received signals. However, this example also illustrated the unnecessarily large orders of the two polynomial matrices generated by the algorithm. By inspection of these matrices, $\underline{\mathbf{D}}_{182}(z)$ (of order 3486) and $\underline{\mathbf{H}}_{182}(z)$ (of order 1733), which are illustrated in Figures 3.3 and 3.4 respectively, their orders are seen to be unnecessarily large, with many of the coefficients associated with the outer lags of both polynomial matrices accounting for a very small proportion of their Frobenius norm.

The SBR2 algorithm was again applied to this example, however, this time implementing the energy based truncation method at the end of each iteration of the algorithm to truncate

the updated polynomial matrix $\underline{\mathbf{D}}_i(z)$ as defined in equation (4.1). This procedure was carried out for the following four specific values of μ: (i) $\mu = 0$, (ii) $\mu = 10^{-10}$, (iii) $\mu = 5 \times 10^{-5}$ and (iv) $\mu = 3 \times 10^{-4}$. In each case, the stopping condition used in Example 3.7 was adopted thus allowing the SBR2 algorithm to continue until all coefficients associated with the off-diagonal polynomial elements of the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$ are less than $\sqrt{\frac{N_1}{4}} \times 10^{-2}$ in magnitude. The results observed for the four cases are presented in Table 4.6, alongside the results obtained when no truncation method is used. The measures used to assess the performance of this truncation method, which are included in the table, are the order of the resulting approximately diagonal polynomial matrix, the relative error of the inverse decomposition denoted as E_{rel}, which will reflect the error encountered due to truncating the transformed polynomial matrix, and also the computational time[4] taken for the SBR2 algorithm to converge. Also recorded in this table are the number of iterations required for the SBR2 algorithm to satisfy the stopping condition for each of the cases. This will often change as a result of truncating the polynomial matrix. Note that the paraunitary transformation matrix $\underline{\mathbf{H}}_i(z)$ is not truncated for these results since it is not necessary to compute this matrix within the iterative routine of the SBR2 algorithm.

	No Truncation	$\mu = 0$	$\mu = 10^{-10}$	$\mu = 5\times10^{-5}$	$\mu = 3\times10^{-4}$
Order of $\underline{\mathbf{D}}_i(z)$	3486	1782	170	42	20
E_{rel}	0	0	7.87×10^{-5}	0.0491	0.1153
Number of Iterations	182	184	184	166	125
Computational Time[2] (Seconds)	39.93	11.99	0.55	0.20	0.12

Table 4.6: Measures to demonstrate the performance of the SBR2 algorithm when applied to the polynomial space-time covariance matrix $\underline{\mathbf{R}}(z)$ from Example 3.7, implementing the energy based truncation method for different values of μ.

The results presented here demonstrate the two main advantages of truncating the polyno-

[4]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

mial matrix $\underline{\mathbf{D}}_i(z)$ using the energy based truncation method; firstly the order of the diagonal matrix obtained by the SBR2 algorithm can be drastically reduced, whilst still maintaining an accurate polynomial matrix decomposition. Even removing only the coefficient matrices positioned in the outer lags with all entires equal to zero (to computational precision) by setting the truncation parameter μ equal to zero, significantly reduces the order of the diagonalised matrix from 3486 to 1782 without compromising the accuracy of the decomposition performed. Furthermore, if μ is set larger than zero, then the order of the matrix can be further reduced, although the transformation performed is no longer norm preserving and will therefore result in some error. Secondly, as a consequence of the reduced order of the polynomial matrix, the computational load and memory storage requirements of the algorithm are reduced and so the computational speed increases. This is clearly demonstrated by the results presented in Table 4.6, illustrating that a suitable choice of μ can be made to optimise the speed and therefore efficiency of the algorithm, whilst also minimising the relative error obtained from the decomposition and the order of the polynomial matrix $\underline{\mathbf{D}}_i(z)$.

It is difficult to know in advance what value to choose for the truncation parameter μ and a suitable choice will depend entirely upon the requirements of the decomposition for the specified application. Clearly, if computational time is not the most important factor when applying the SBR2 algorithm, it is better to set the truncation parameter to a very small value, for example $\mu = 10^{-10}$, or equal to zero, as this will minimise the relative error of the decomposition. If the order of the resulting diagonal polynomial matrix is then too large for the application of the decomposition, the fixed bound truncation method can be applied to reduce the order to the required value. This particular problem is often encountered when applying the algorithm in applications relating to MIMO communication systems as discussed further in the penultimate chapter of this thesis. For example, if the polynomial matrix $\underline{\mathbf{D}}_{184}(z) \in \mathbb{C}^{4\times4\times170}$ obtained using $\mu = 10^{-10}$ is truncated to be of order 20, the relative error of the decomposition is found to be 0.0542. This is nearly half of the value obtained when truncating the order of the polynomial matrix throughout each iteration of the algorithm with a larger value of μ, but obtaining the same final order. This can be seen in Table 4.6 for the case $\mu = 3 \times 10^{-4}$.

Figure 4.8 shows how the order of the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$ increases at each iteration i of the SBR2 algorithm, for each of the five cases of truncation recorded in Table 4.6. This figure clearly illustrates the steady increase in the order of the polynomial matrix when no truncation method is used, leading to a matrix of a very large order. Furthermore, by comparing the individual plots for each truncation value with the relative errors obtained for each case, it is clear that the order can be vastly reduced whilst still maintaining an accurate level of decomposition.

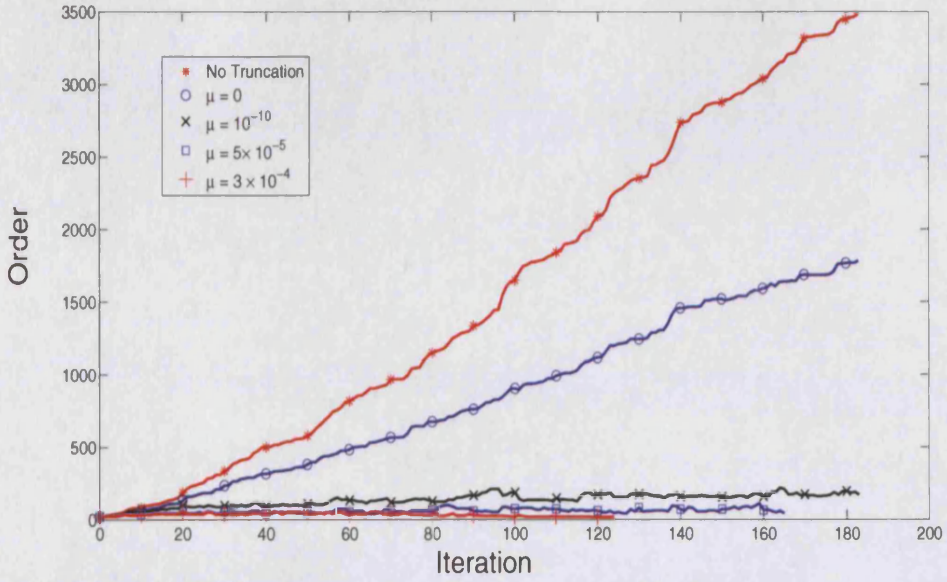

Figure 4.8: The order of the polynomial matrix $\underline{\mathbf{D}}_i(z)$ at the end of each iteration i of the SBR2 algorithm for the cases when (i) no truncation method is used, and then the energy based truncation method is applied to the transformed matrix $\underline{\mathbf{D}}_i(z)$ with (ii) $\mu = 0$, (iii) $\mu = 10^{-10}$, (iv) $\mu = 5 \times 10^{-5}$ and (v) $\mu = 3 \times 10^{-4}$.

Finally, the paraunitary transformation matrix, which can also grow to an excessive order as illustrated in Figure 3.4, was also truncated using the method suitable for non para-Hermitian polynomial matrices described in Section 4.3.2, with the truncation parameter μ set as 5×10^{-5}. Truncating both the polynomial transformation matrix $\underline{\mathbf{H}}_i(z)$ and the transformed polynomial matrix $\underline{\mathbf{D}}_i(z)$ at the end of each iteration of the algorithm using this value of μ (although they can be set differently), the relative error of the polynomial matrix decomposition increased from 0.0491 to 0.0495. Figures 4.9 and 4.10 illustrate the resulting

diagonal matrix $\underline{\mathbf{D}}(z)$ and paraunitary transformation matrix $\underline{\mathbf{H}}(z)$ obtained for this instance.

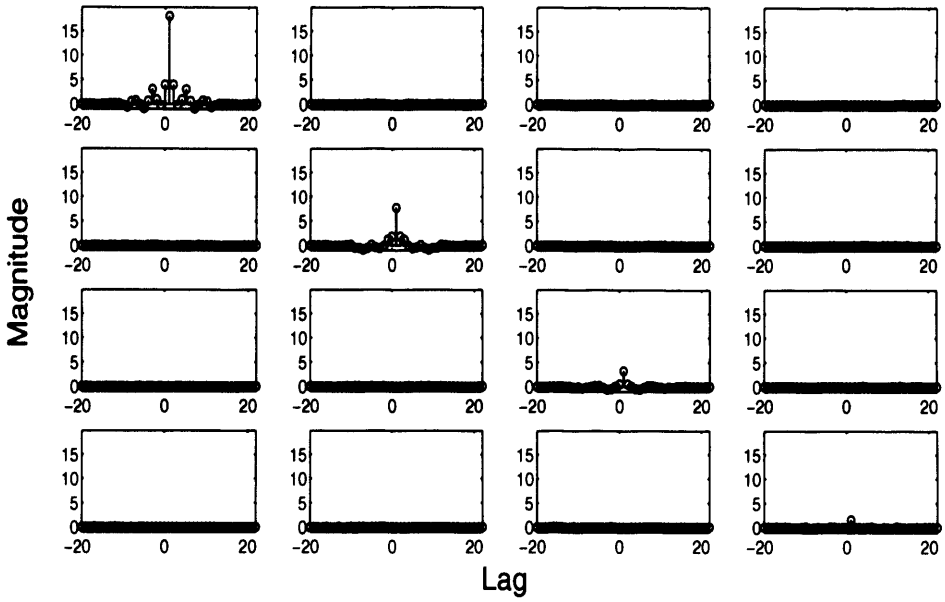

Figure 4.9: The diagonal matrix $\underline{\mathbf{D}}(z)$ produced by applying the SBR2 algorithm to the polynomial space-time covariance matrix $\underline{\mathbf{R}}(z)$ from Example 3.7, implementing the energy based truncation method with $\mu = 5 \times 10^{-5}$.

4.7 Conclusions

This chapter has presented two polynomial matrix truncation methods, both of which can be used within the SBR2 algorithm to stop the order of both the paraunitary transformation matrix and the resulting diagonal polynomial matrix becoming unnecessarily large. The results have clearly demonstrated that the energy based truncation method is the best method to use, provided a suitable choice of the truncation parameter μ has been made, as it allows some control over the accuracy of the overall matrix decomposition. The truncation method can be applied to both of the polynomial matrices $\underline{\mathbf{H}}_i(z)$ and $\underline{\mathbf{D}}_i(z)$ from equation (4.1) at the end of each iteration of the algorithm, which will consequently reduce the computational load, enabling the computational time of the SBR2 algorithm to be drastically reduced. Using the truncation method does not affect the proof of convergence of the SBR2 algorithm.

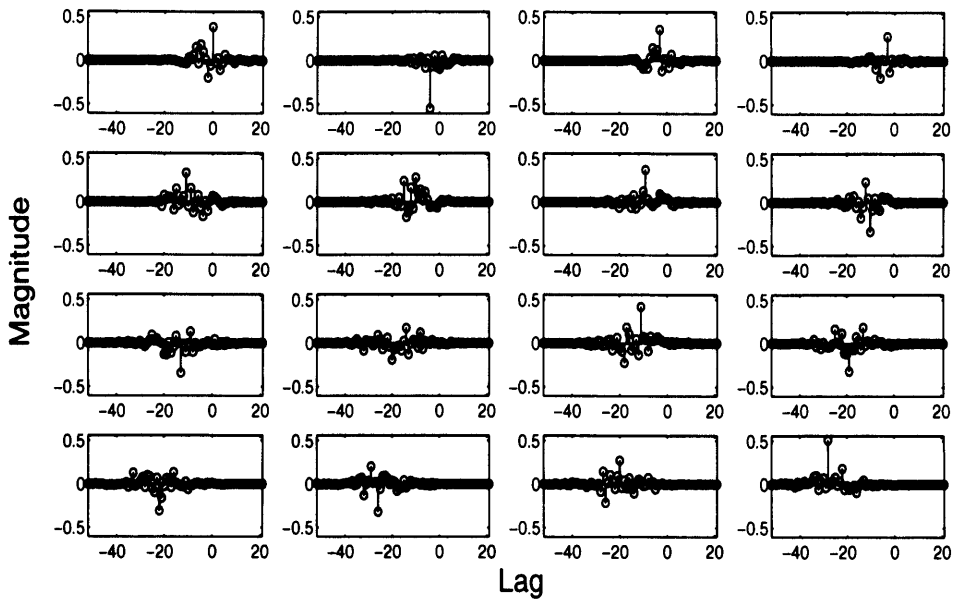

Figure 4.10: The paraunitary matrix $\underline{\mathbf{H}}(z)$ produced by applying the SBR2 algorithm to the polynomial space-time covariance matrix $\underline{\mathbf{R}}(z)$ from Example 3.7, using the energy based truncation method for non para-Hermitian matrices with $\mu = 5 \times 10^{-5}$.

Furthermore, using a suitable value of the truncation parameter μ will also result in a diagonal polynomial matrix of a much smaller order, as is often required in practice, e.g. when applying the decomposition to MIMO communications. This application is discussed further, together with a brief description of how the relative error affects the overall performance in the penultimate chapter of this thesis.

The three main objectives of the truncation method are

1. To reduce the order of the output matrices $\underline{\mathbf{D}}(z)$ and $\underline{\mathbf{H}}(z)$ obtained by the SBR2 algorithm, which were previously unnecessarily large,

2. to reduce the computational time taken to calculate the decomposition, whilst

3. not compromising the accuracy of the decomposition performed significantly.

Unfortunately, finding the appropriate value of μ involves a process of trial and error, trying different values of μ until one is found, which leads to a sufficiently accurate polynomial matrix decomposition, whilst optimising the computational speed and producing matrices of

a sufficiently small order. At the very least, the energy based truncation method should be applied to both the paraunitary transformation matrix $\underline{\mathbf{H}}_i(z)$ and the transformed matrix $\underline{\mathbf{D}}_i(z)$, at the end of each iteration of the algorithm with the truncation parameter μ set equal to zero. As demonstrated by the results in this chapter, this can often drastically reduce the order of the two matrices, thereby allowing the computational load of the algorithm to be reduced and the computational speed to increase. Furthermore, using this value of μ will not compromise the accuracy of the decomposition performed and the transformation matrix will still be paraunitary. If a particular order is required for the resulting polynomial matrices and the computational time is not the main concern of the algorithm, then it is better to set μ equal to zero and then truncate the orders of the resulting polynomial matrices obtained from this decomposition again using the fixed bound method to obtain the appropriate orders. This will ensure a more accurate level of polynomial matrix decomposition than that obtained when truncating at each iteration with $\mu > 0$. The observation is made here as this technique is used for results presented in the remaining chapters of this thesis.

The order of the matrices can be reduced further by setting $\mu > 0$, but then this will affect the accuracy of the decomposition as the transformation will no longer be norm (or energy) preserving. Moreover, if it is used to truncate the transformation matrix $\underline{\mathbf{H}}_i(z)$, then this matrix will no longer be exactly paraunitary. For this reason, when setting $\mu > 0$ it is important to always check the relative error of the decomposition and also the proportion of the Frobenius norm of both the polynomial matrices $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{H}}(z)$ that has been lost due to truncation. If the truncation parameter is suitably chosen the technique can be implemented to optimise the speed of the algorithm, without significantly compromising the accuracy of the decomposition. The appropriate choice for the truncation parameter μ for each application of the decomposition, in terms of optimising the speed and minimising the relative error of the decomposition will have to found experimentally for each matrix to which the SBR2 algorithm is to be applied.

Finally, the truncation methods presented here, can also be applied to any polynomial matrices, not just those calculated within the SBR2 algorithm. The energy based bound is also used with the other polynomial matrix decompositions to be presented in this thesis.

The computational complexity of the SBR2 algorithm is presented in Appendix C. Note that the polynomial matrix truncation method has been discussed in [59], where the energy based truncation method is used within the SBR2 algorithm to obtain an accurate PEVD.

Chapter 5

The QR Decomposition of a Polynomial Matrix

5.1 Introduction

The Polynomial matrix QR Decomposition (PQRD) is a technique for factorising a polynomial matrix into an upper triangular and a paraunitary polynomial matrix and can be applied to either a square or rectangular polynomial matrix, where the coefficients of each polynomial element can be either real or complex. For a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$, the objective of the PQRD is to calculate a paraunitary polynomial matrix $\underline{\mathbf{Q}}(z) \in \underline{\mathbb{C}}^{p\times p}$ such that

$$\underline{\mathbf{Q}}(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{R}}(z) \tag{5.1}$$

where $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times q}$ is an upper triangular polynomial matrix. This clearly constitutes a generalisation of the QR decomposition (QRD) from matrices with scalar elements to those with polynomial elements, each with an associated set of coefficients. Note that unlike the PEVD, which requires the input matrix to be a para-Hermitian polynomial matrix, there is no special structure or requirements of the polynomial matrix $\underline{\mathbf{A}}(z)$ for the QRD.

Calculating the QRD of a polynomial matrix is clearly a more complex problem than formulating the same decomposition of a scalar matrix, as each element of the matrix $\underline{\mathbf{A}}(z)$ now consists of a series of polynomial coefficients. In order to drive one element of the matrix to zero, all coefficients of this element must be driven to zero and this can no longer be achieved using only Givens rotations [6]. Instead, a similar approach is implemented to that used when generating the paraunitary transformation matrix required within the

SBR2 algorithm and so the paraunitary polynomial matrix $\underline{\mathbf{Q}}(z)$ is formulated as a series of elementary rotation matrices interspersed with delay matrices.

This chapter introduces three different algorithms for calculating a QR decomposition of a polynomial matrix, where each of the algorithms adopts a slightly different technique for formulating the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$. The three algorithms are defined as

1. PQRD By Steps (PQRD-BS)
2. PQRD By Columns (PQRD-BC)
3. PQRD by Sequential Best Rotation (PQRD-SBR)

As with the SBR2 algorithm for calculating the PEVD, the QR decomposition of a polynomial matrix is not unique and so the three decomposition algorithms above will not necessarily generate exactly the same paraunitary or upper triangular polynomial matrices when given the same input matrix $\underline{\mathbf{A}}(z)$; this point is considered in Section 5.7 of this chapter. Convergence of each of the algorithms is then discussed, before each technique is demonstrated by applying the algorithm to a simple polynomial matrix. A possible application of the decomposition is in MIMO communications, where it is often required to reconstruct data sequences that have been distorted due to the effects of multipath propagation, leading to intersymbol interference (ISI) and co-channel interference (CCI). This application of the PQRD is discussed towards the end of this thesis in Chapter 8. Before discussing each of the three algorithms for calculating the PQRD, the concept of a polynomial Givens rotation is introduced.

5.2 An Elementary Polynomial Givens Rotation

An elementary polynomial Givens rotation (EPGR) is a polynomial matrix that can be applied to either a polynomial vector or matrix to selectively zero one coefficient of a polynomial element. Firstly, we discuss how this matrix can be applied, in the simplest case, to a polynomial vector $\underline{\mathbf{a}}(z) \in \underline{\mathbb{C}}^{2\times 1}$ and then subsequently discuss how it can be applied to a polynomial matrix.

An EPGR takes the form of a Givens rotation preceded by an elementary time shift matrix. For example, a 2×2 EPGR is formulated as follows

$$\underline{\widehat{\mathbf{G}}}^{(\alpha,\theta,\phi,t)}(z) = \begin{pmatrix} ce^{i\alpha} & se^{i\phi} \\ -se^{-i\phi} & ce^{-i\alpha} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & z^t \end{pmatrix} \tag{5.2}$$

$$= \begin{pmatrix} ce^{i\alpha} & se^{i\phi}z^t \\ -se^{-i\phi} & ce^{-i\alpha}z^t \end{pmatrix} \tag{5.3}$$

where c and s define the cosine and sine of the angle θ respectively. The aim of this matrix, when applied to a polynomial vector $\underline{\mathbf{a}}(z) \in \underline{\mathbb{C}}^{2\times 1}$ as demonstrated

$$\begin{pmatrix} ce^{i\alpha} & se^{i\phi}z^t \\ -se^{-i\phi} & ce^{-i\alpha}z^t \end{pmatrix} \begin{pmatrix} \underline{a}_1(z) \\ \underline{a}_2(z) \end{pmatrix} = \begin{pmatrix} \underline{a}'_1(z) \\ \underline{a}'_2(z) \end{pmatrix} \tag{5.4}$$

is to drive a specified coefficient from the polynomial vector $\underline{\mathbf{a}}(z)$ to zero. For example, to zero the coefficient $a_2(\tau)$, the lag parameter in the EPGR is set as $t = \tau$ and the rotation angles are chosen such that

$$\tan(\theta) = \frac{|a_2(t)|}{|a_1(0)|}, \tag{5.5}$$

$$\phi = -arg(a_2(t)) \tag{5.6}$$

and

$$\alpha = -arg(a_1(0)) \tag{5.7}$$

thus resulting in $a'_2(0) = 0$. Furthermore, following the application of the EPGR the coefficient $a'_1(0)$ is real and $|a'_1(0)|^2 = |a_1(0)|^2 + |a_2(t)|^2$. Note that if $|a_1(0)| = 0$ in equation (5.5), then set $\theta = \pi/2$. The rotation angles ϕ and α could have alternatively been chosen as

$$\phi = arg(a_1(0)) \quad \text{and} \quad \alpha = arg(a_2(t)), \tag{5.8}$$

however this choice of angles will not ensure the resulting coefficient $a_1'(0)$ is a positive real scalar, which is required for uniqueness in the scalar matrix QRD. Note that unlike the rotation angles required for the SBR2 algorithm, equation (5.5) will not have multiple solutions and so it does not matter whether the basic or four quadrant arctangent function is used for its calculation.

This technique can now easily be extended to formulate an EPGR, which can be applied to a polynomial matrix, $\underline{\mathbf{A}}(z) \in \in \underline{\mathbb{C}}^{p\times q}$, to drive a particular coefficient of one of the polynomial elements of this matrix to zero. For example, the EPGR required to zero the coefficient $a_{jk}(\tau)$ of the polynomial element $\underline{a}_{jk}(z)$ takes the form of a $p \times p$ identity matrix with the exception of the four elements situated at the intersection of rows j and k with columns j and k. These elements are given by the 2×2 EPGR matrix $\widehat{\underline{\mathbf{G}}}^{(\alpha,\theta,\phi,t)}(z)$ described by equation (5.3), where the lag parameter is set as $t = \tau$ and the coefficients required for calculating the rotation angles in equations (5.5) - (5.7) are now

$$a_2(\tau) = a_{jk}(\tau) \quad \text{and} \quad a_1(0) = a_{kk}(0). \tag{5.9}$$

This $p \times p$ EPGR matrix will be defined as $\widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ where the superscripts j and k have been added to denote the position of the polynomial coefficient that the EPGR is designed to zero and can be applied to $\underline{\mathbf{A}}(z)$ to obtain the transformed matrix

$$\underline{\mathbf{A}}'(z) = \widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z) \tag{5.10}$$

where as a result of the application of the EPGR $a'_{jk}(0) = 0$, $a'_{kk}(0)$ is real and $|a'_{kk}(0)|^2 = |a_{kk}(0)|^2 + |a_{jk}(\tau)|^2$. The effect of this transformation is to shift the coefficient $a_{jk}(\tau)$ so that it becomes the coefficient of z^0 in the same polynomial element, i.e. it becomes the coefficient $a_{jk}(0)$, and then apply the appropriate rotation so that the coefficient is forced to equal to zero.

Note that an EPGR matrix is paraunitary by construction as each component of the matrix, i.e. the Givens rotation and the elementary time shift matrix, are both paraunitary.

As a consequence of this, the transformation is norm preserving and so

$$\left\| \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z) \right\|_F = \|\underline{\mathbf{A}}(z)\|_F . \tag{5.11}$$

5.3 Complete Polynomial Givens Rotation

The objective of a complete polynomial Givens rotation (CPGR) is to zero an entire polynomial element of a polynomial vector or matrix by driving each coefficient of a polynomial element to zero in turn. In practice, as with the SBR2 algorithm, it is often not feasible to zero all coefficients of a polynomial element and so it is only required that each coefficient of the polynomial element becomes sufficiently small. A CPGR will firstly be introduced in its simplest case as a 2×2 matrix applicable to a polynomial vector $\underline{\mathbf{a}}(z) \in \underline{\mathbb{C}}^{2\times 1}$, before extending the concept so it can be applied to a polynomial matrix of any dimension.

A series of EPGRs can be applied iteratively to the polynomial vector $\underline{\mathbf{a}}(z) \in \underline{\mathbb{C}}^{2\times 1}$, as demonstrated by equation (5.4), to drive all coefficients of a specified polynomial element arbitrarily close to zero. For example to drive the polynomial element $\underline{a}_2(z)$ to zero, at each iteration the rotation angles θ, ϕ and α and the lag parameter t are chosen to zero the coefficient within $\underline{a}_2(z)$ with maximum magnitude, this coefficient will be referred to as the dominant coefficient. If this coefficient is not unique, then any of the dominant coefficients from the polynomial element may be chosen. The complete series of EPGRs required, constitutes a CPGR, which will be denoted by the matrix $\underline{\mathbf{G}}^{(2,1)}(z)$, where the superscripts denote the row and column position of the polynomial element that the matrix is attempting to annihilate. A matrix of this form can be applied to a polynomial vector $\underline{\mathbf{a}}(z) \in \underline{\mathbb{C}}^{2\times 1}$ such that

$$\underline{\mathbf{G}}^{(2,1)}(z) \begin{bmatrix} \underline{a}_1(z) \\ \underline{a}_2(z) \end{bmatrix} \cong \begin{bmatrix} \underline{a}'_1(z) \\ 0 \end{bmatrix} \tag{5.12}$$

where all coefficients of the polynomial element $\underline{a}_2(z)$ have been driven arbitrarily close to zero over a series of EPGRs. In practice it is often not feasible to zero all coefficients of a polynomial element, hence the approximate equality in equation (5.12). Instead the coef-

ficients are driven to zero until the magnitude of all coefficients of the polynomial element $\underline{a}_2(z)$ are sufficiently small and the following stopping condition is satisfied

$$|a_2(t)| < \epsilon \tag{5.13}$$

$\forall t \in \mathbb{Z}$ and where $\epsilon > 0$ is some pre-specified small value.

Again as with the EPGR, this idea is easily extended to a general CPGR that can be applied to a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ to drive all coefficients of one of the polynomial elements sufficiently small. At each iteration a $p\times p$ EPGR matrix is applied to the polynomial matrix to zero the dominant coefficient of a specified polynomial element and this process is repeated until all coefficients of the polynomial element are suitably small and satisfy a similar stopping condition to that demonstrated in equation (5.13). For example, to zero the polynomial element $\underline{a}_{jk}(z)$, the required CPGR matrix is calculated as the series of EPGRs

$$\underline{\mathbf{G}}^{(j,k)}(z) = \underline{\widehat{\mathbf{G}}}_i^{(j,k,\alpha_i\theta_i,\phi_i,t_i)}(z)\ldots\underline{\widehat{\mathbf{G}}}_1^{(j,k,\alpha_1\theta_1,\phi_1,t_1)}(z) \tag{5.14}$$

where i is the number of EPGRs required to drive all coefficients of the polynomial element sufficiently close to zero and the variables α_i, θ_i, ϕ_i and t_i denote the rotation and lag parameters required to zero the i^{th} dominant coefficient respectively. Note that each of the EPGRs is paraunitary and so the complete polynomial Givens rotation will also be paraunitary. Furthermore, this paraunitary transformation is also norm preserving and so $\|\underline{\mathbf{A}}(z)\|_F^2 = \left\|\underline{\mathbf{G}}^{(j,k)}(z)\underline{\mathbf{A}}(z)\right\|_F^2$. A proof of convergence for a complete polynomial Givens rotation step can now be easily deduced.

5.3.1 Convergence of a CPGR

With every application of an EPGR, to zero the dominant coefficient $a_{jk}(t)$, the quantity $|a_{kk}(0)|^2$ will increase monotonically by the magnitude squared of the largest coefficient within the element, i.e. by the quantity $|a_{jk}(t)|^2$. Furthermore, this quantity is bounded above by the squared Frobenius norm of the k^{th} column of $\underline{\mathbf{A}}(z)$, i.e. the quantity $\sum_{\tau}\sum_{i=1}^{p}|a_{ik}(\tau)|^2$, which

will remain constant throughout all iterations of the CPGR. As $|a_{kk}(0)|^2$ is monotonically increasing and bounded above, over a series of EPGRs the stopping condition similar to that demonstrated in equation (5.13) is guaranteed and the complete polynomial Givens rotation converges in this respect.

5.4 Algorithm 1: PQRD by Steps

This section describes the first of the three algorithms for calculating the PQRD and directly extends the QR decomposition from matrices with complex scalar elements to polynomial matrices, where each element is now a polynomial with an associated set of complex coefficients. The algorithm proceeds to perform the decomposition by following the same strategy in eliminating the entries of the matrix as is used in the Givens method for achieving the QRD of a scalar matrix, i.e. by driving the elements beneath the diagonal to zero in a specified order. There are several different orderings that can be implemented to obtain the scalar matrix QRD, but for the purposes of this thesis, it is assumed that the elements are eliminated starting with the uppermost left element beneath the diagonal of the matrix and then proceed through all elements beneath the diagonal in each row from left to right, before moving to the next row down and so on. However, as each element of the matrix is now a polynomial, all coefficients of the polynomial element must be eliminated to ensure that each polynomial element is approximately equal to zero. This can no longer be achieved by applying a scalar Givens rotation matrix; instead a CPGR is required.

5.4.1 The PQRD by Steps Algorithm

The PQRD by Steps algorithm aims to transform a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ into an approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times q}$ by means of a paraunitary transformation matrix $\underline{\mathbf{Q}}(z) \in \underline{\mathbb{C}}^{p\times p}$, as demonstrated

$$\underline{\mathbf{Q}}(z)\underline{\mathbf{A}}(z) \cong \underline{\mathbf{R}}(z). \tag{5.15}$$

The paraunitary matrix is calculated within the algorithm as a series of ordered steps, where at each step all coefficients associated with one polynomial element situated beneath the diagonal of the polynomial matrix $\underline{\mathbf{A}}(z)$ are driven sufficiently small by applying the appropriate CPGR. For example, the transformation observed at one step, say the i^{th} step, to drive all coefficients of the polynomial element $\underline{a}_{jk}(z)$ sufficiently small, is therefore of the form

$$\underline{\mathbf{A}}_i(z) = \underline{\mathbf{G}}^{(j,k)}(z)\underline{\mathbf{A}}_{i-1}(z) \tag{5.16}$$

where $\underline{\mathbf{G}}^{(j,k)}(z)$ is the complete polynomial Givens rotation designed to drive all coefficients of the polynomial element $\underline{a}_{jk}(z)$ sufficiently small.

Each step of the algorithm, however, operates as an iterative process, where at each iteration an EPGR matrix is applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ to zero the dominant coefficient of the $(j,k)^{th}$ polynomial element. For example, if the dominant coefficient is found to be $a_{jk}(t)$, the transformation is of the form

$$\underline{\mathbf{A}}^{'}(z) = \widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z), \tag{5.17}$$

where $\widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ defines the EPGR formulated according to Section 5.2 to zero the dominant coefficient $a_{jk}(t)$. Note that if the dominant coefficient is not unique then any of the dominant coefficients from the polynomial element may be chosen. The effect of this transformation is to firstly shift the dominant coefficient onto the coefficient matrix of order zero, $\mathbf{A}(0)$, and then apply the appropriate rotation so that the coefficient becomes equal to zero, i.e. $a'_{jk}(0) = 0$. Furthermore, the transformation performed by the EPGR to drive the dominant coefficient $a_{jk}(\tau)$ to zero, will result in $|a'_{kk}(0)|^2 = |a_{jk}(\tau)|^2 + |a_{kk}(0)|^2$ and $a'_{kk}(0)$ will be real. Note that also as a result of the transformation all coefficients in the k^{th} row of the polynomial matrix $\underline{\mathbf{A}}(z)$ have been shifted, which is caused by the application of the elementary delay matrix incorporated in the EPGR. In addition all coefficients in the j^{th} and k^{th} rows of the matrix $\underline{\mathbf{A}}(z)$ have changed due to the rotation. Elements in all other rows of the matrix, excluding rows j and k, are unaffected by the application of the EPGR.

This iterative process is repeated until all coefficients from the polynomial element $\underline{a}_{jk}(z)$

are sufficiently small and satisfy the stopping condition

$$|a_{jk}(\tau)| < \epsilon \tag{5.18}$$

$\forall \tau \in \mathbb{Z}$ where $\epsilon > 0$ is a pre-specified small value. As explained previously, in practice it is often not feasible to drive all coefficients of a polynomial element to zero by application of a CPGR and so instead it is only ever required that a stopping condition such as that demonstrated by (5.18) is satisfied.

This completes one step of the PQRD-BS algorithm. To begin the subsequent step, $\underline{\mathbf{A}}(z)$ is replaced with $\underline{\mathbf{A}}'(z)$ and the indices j and k are amended appropriately, moving to the next polynomial element in the ordering. The ordering begins with the uppermost left element beneath the diagonal, $\underline{a}_{21}(z)$, and then moves across all elements beneath the diagonal in each row, before moving to the next row down and repeating the process. This continues until the algorithm has visited all elements beneath the diagonal of the polynomial matrix $\underline{\mathbf{A}}(z)$. Once all elements beneath the diagonal have been visited, one sweep of the algorithm has been completed. Each sweep of the PQRD-BS algorithm operates in a finite number of steps, equal to the cardinality of the set of elements beneath the diagonal of the polynomial matrix $\underline{\mathbf{A}}(z)$. However, each step operates as an iterative process and the number of iterations required in a single step cannot be predetermined. Note that multiple sweeps of the algorithm may be required and this point is discussed further in Section 5.4.2.

Finally, following i steps of the algorithm, the transformation performed is of the form

$$\underline{\mathbf{A}}_i(z) = \underline{\mathbf{Q}}_i(z)\underline{\mathbf{A}}(z) \tag{5.19}$$

where $\underline{\mathbf{Q}}_i(z)$ is the product of i CPGR matrices and will be paraunitary by construction. Furthermore, following all steps of the first sweep of the algorithm, the matrix decomposition performed can be expressed as

$$\underline{\mathbf{A}}_T(z) = \underbrace{\underline{\mathbf{G}}^{(p,N)}(z)\ldots\underline{\mathbf{G}}^{(2,1)}(z)}_{=\,\underline{\mathbf{Q}}_T(z)}\underline{\mathbf{A}}(z) \tag{5.20}$$

where T defines the number of elements beneath the diagonal of $\underline{\mathbf{A}}(z)$, $\underline{\mathbf{G}}^{(j,k)}(z)$, for $j = 2,\ldots,p$ and $k = 1,\ldots,N$, denotes the CPGR designed to drive all coefficients of the polynomial element $\underline{a}_{jk}(z)$ sufficiently small and where

$$N = \begin{cases} q & \text{if } p > q \\ p-1 & \text{if } p \leq q \end{cases}. \tag{5.21}$$

Note that in the degenerate case, where the order of the polynomial matrix $\underline{\mathbf{A}}(z)$ is zero, this algorithm simply reduces to computing the QR decomposition of a scalar matrix by applying an ordered series of Givens rotations. Of the three algorithms presented in this chapter this is the most similar to the conventional technique for computing the QRD of a scalar matrix. A concise description of the PQRD-BS algorithm can be found in Appendix B.

5.4.2 Multiple Sweeps

The various techniques for calculating the QRD of a scalar matrix, in particular the Givens method, will ensure that all elements beneath the diagonal of the matrix are driven to zero and once an element is equal to zero it remain so through future applications of Givens rotations, due to the order in which elements are eliminated. However, this is not possible when formulating the QRD of a polynomial matrix. Although the PQRD-BS algorithm drives the dominant coefficient at each iteration of each step to zero, it only ensures that all coefficients of a polynomial element are suitably small, according to the stopping condition demonstrated by equation (5.18), before beginning the subsequent step and moving to the next polynomial element in the ordering. Through future steps and therefore rotations of the algorithm, these small coefficients could be rotated with other suitably small coefficients, causing them to increase in magnitude and possibly violate the stopping criterion ϵ. For this reason, multiple sweeps of the algorithm may be required to ensure that all coefficients relating to polynomial elements beneath the diagonal of $\underline{\mathbf{A}}'(z)$ are less than ϵ in magnitude.

Note that when completing the step to drive all coefficients of the polynomial element

$\underline{a}_{jk}(z)$ sufficiently small, the quantity $\sum_{t} \sum_{m=n+1}^{p} |a_{mn}(t)|^2$ will remain constant for all values of $n = 1, \ldots, k-1$ and for all future steps in that particular sweep of the algorithm. Therefore, any coefficients positioned in columns to the left of the polynomial element $\underline{a}_{jk}(z)$, which are affected by the rotations and as a result have increased in magnitude to be larger than ϵ, will be bounded above by the Frobenius norm of the elements beneath the diagonal in the same column of the matrix.

5.4.3 Convergence of the PQRD by Steps Algorithm

Completion of the scalar matrix QRD is easily deduced as the Givens method will zero all elements of the matrix beneath the diagonal of the matrix exactly and once an element is equal to zero it remains so through future steps of the algorithm. However, this cannot be achieved with the method for calculating the QR decomposition of a polynomial matrix discussed here. Although the dominant coefficient of a polynomial element is driven to zero at each iteration, the algorithm only ensures that all coefficients of an element are suitably small before moving to the next polynomial element in the ordering. Therefore, through future rotations of the algorithm, these small coefficients could be rotated with other suitably small coefficients, causing them to increase in magnitude and perhaps exceed the stopping criterion ϵ. For this reason a proof of convergence for the algorithm does not simply follow from the proof of convergence given for one polynomial element, detailed in Section 5.3.1.

Before discussing the convergence of the algorithm, three measures on the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$ must be introduced which will be used within the proof of convergence for the PQRD-BS algorithm. The measures are defined as follows

$$E_{ij} = \sum_{\tau} |a_{ij}(\tau)|^2, \tag{5.22}$$

$$E_{Lj} = \sum_{\tau} \sum_{i=j+1}^{p} |a_{ij}(\tau)|^2 \tag{5.23}$$

and

$$E_{Uj} = \sum_{\tau} \sum_{i=1}^{j-1} |a_{ij}(\tau)|^2 . \tag{5.24}$$

These measures define the squared Frobenius norm of the polynomial element $\underline{a}_{ij}(z)$, the squared Frobenius norm of all elements beneath the diagonal in the j^{th} column of $\underline{\mathbf{A}}(z)$ and the squared Frobenius norm of all elements above the diagonal in the j^{th} column of $\underline{\mathbf{A}}(z)$ respectively. As discussed previously, the squared Frobenius norm of a polynomial element will also be referred to as the energy of that element. If any of the three expressions above are followed by the further notation (0), e.g. $E_{Lj}(0)$, this denotes the appropriate expression evaluated only on the coefficient plane of order zero (i.e. set $\tau = 0$). These measures will also be required within the proofs of convergence of the two subsequent PQRD algorithms to be discussed later in this chapter.

Whatever the size or dimension of the polynomial matrix $\underline{\mathbf{A}}(z)$, the first polynomial element to be driven to zero in the ordering will always be $\underline{a}_{21}(z)$. In driving all coefficients of this polynomial element sufficiently small, the quantity $E_{11}(0)$ will increase monotonically. It will increase at each iteration of this step by the magnitude squared of the dominant coefficient of the polynomial element $\underline{a}_{21}(z)$, until the magnitude of the dominant coefficient falls less than a pre-specified small value $\epsilon > 0$ according to the stopping condition (5.18). Let the dominant coefficient at each iteration be denoted as g, then at each iteration $E_{11}(0)$ will increase by g^2. Subsequent steps of the algorithm, which aim to drive other polynomial elements situated anywhere beneath the diagonal of the matrix $\underline{\mathbf{A}}(z)$ to zero, can never allow this quantity to decrease, even if a subsequent step requires a CPGR to be applied to a column positioned to the right. However, future rotations can affect coefficients beneath the diagonal, allowing previously sufficiently small coefficients to increase in magnitude and possibly exceed ϵ. Furthermore, any steps to drive any polynomial element in the first column to zero, will lead to a further increase in the quantity $E_{11}(0)$ and as a consequence, this quantity will increase montonically throughout all steps of the algorithm. This quantity is also bounded above by the total energy in the first column, which will be denoted as $E_1 = E_{11} + E_{L1} + E_{U1}$, and so it must have a supremum s_1. It follows that for any $\delta > 0$ there must be an iteration L_1, beyond which $|s_1 - E_{11}(0)| < \delta$. Any subsequent step must then satisfy $g^2 \leq |s_1 - E_{11}(0)| < \delta$ and

so there exists an iteration at which the magnitude squared of the maximum coefficient in $\underline{a}_{21}(z)$ is bounded by δ.

Similarly when the ordering reaches the polynomial element $\underline{a}_{jk}(z)$, the series of EPGRs required to drive this element sufficiently close to zero, will cause the quantity $E_{kk}(0)$ to increase monotonically. At each iteration this quantity will increase by the magnitude squared of the dominant coefficient within the polynomial element $\underline{a}_{jk}(z)$. Subsequent iterations of the algorithm will never allow this quantity to decrease due to the order in which the polynomial elements are driven sufficiently small. The proof of convergence follows directly from the proof of convergence given for the polynomial element $\underline{a}_{21}(z)$ and so there exists an iteration by which the magnitude squared of every coefficient in the element is less than a certain value.

During the first sweep of the algorithm coefficients previously guaranteed to converge can be rotated at a later step, possibly causing them to increase in magnitude and become larger than ϵ. If this happens then a second sweep of the algorithm is undertaken. Note that the quantity $E_{11}(0)$ has only ever increased and has not been affected by any rotations or delays applied to elements positioned in columns to the right of it. Every step involving elements beneath the diagonal of this column will further force $E_{11}(0)$ to increase, which can clearly not continue indefinitely. Whatever happens to elements in columns to the right, there will come a point where no further rotations are required in the first column. Once this point has been reached, the quantity $E_{22}(0)$ will increase monotonically through all future steps until a point is acheived where this quantity can no longer increase and further rotations are not required in this column. This will continue to happen to all diagonal elements situated on the coefficient plane of order zero working from left to right through the matrix. The final quantity to increase monotonically will be $E_{qq}(0)$ if $p \geq q$, otherwise $E_{pp}(0)$ if $p < q$, and once no further rotations can be applied, the algorithm has converged.

As with the proof of convergence for the SBR2 algorithm, using the truncation method does not affect this proof of convergence.

5.5 Algorithm 2: PQRD by Columns

The second algorithm for calculating the PQRD of a polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ according to equation (5.15) is now introduced. This algorithm operates in a similar way to the PQRD-BS algorithm, by calculating the paraunitary polynomial matrix $\underline{\mathbf{Q}}(z) \in \underline{\mathbb{C}}^{p\times p}$ as a series of ordered steps. However, each step of this algorithm consists of an iterative process to drive the coefficients associated with all polynomial elements situated beneath the diagonal of a particular column of the polynomial matrix $\underline{\mathbf{A}}(z)$ sufficiently small, which is achieved by applying a series of EPGRs. The PQRD By Columns (PQRD-BC) algorithm is a natural progression from the PQRD-BS algorithm discussed in Section 5.4 and the difference between the process of steps for the two algorithms operate is illustrated in Figure 5.1. Before discussing the possible advantages of this method, a detailed description of the algorithm is firstly discussed.

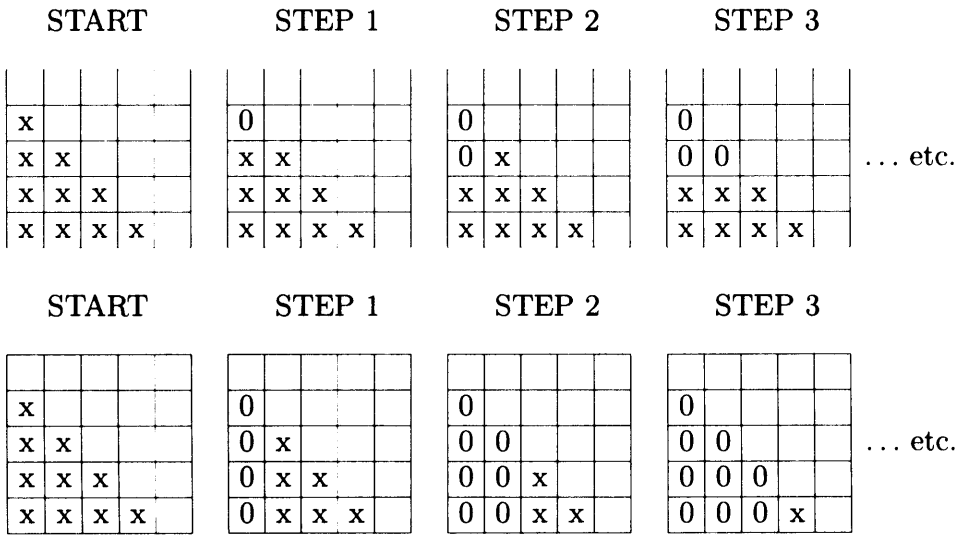

Figure 5.1: Diagram to illustrate the different orders in which the PQRD-BS (top) and the PQRD-BC (bottom) algorithms zero polynomial elements of an 5×5 polynomial matrix.

5.5.1 The PQRD by Columns Algorithm

The algorithm operates as a series of steps, where at each step all coefficients associated with the polynomial elements positioned beneath the diagonal of one column of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ are driven sufficiently small. The algorithm begins the first step with the first column of the matrix.

The first step begins by locating the dominant coefficient positioned beneath the diagonal of the first column of the input matrix $\underline{\mathbf{A}}(z)$. As with the PQRD-BS algorithm, if the dominant coefficient is not unique then any of the dominant coefficients may be chosen. Assume that this coefficient is found to be $a_{j1}(\tau)$, which denotes the coefficient of $z^{-\tau}$ in the polynomial element $\underline{a}_{j1}(z)$, where clearly the row index $j > 1$.

The appropriate EPGR matrix, $\underline{\widehat{\mathbf{G}}}^{(j,1,\alpha,\theta,\phi,t)}(z)$, is then formulated according to Section 5.2, where the lag parameter is set as $t = \tau$ and the rotation parameters α, θ are ϕ are calculated according to equations (5.5) - (5.7), where the coefficients required for these calculations now correspond to $a_2(t) = a_{j1}(t)$ and $a_1(0) = a_{11}(0)$. The EPGR is then applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ to obtain the transformed polynomial matrix

$$\underline{\mathbf{A}}'(z) = \underline{\widehat{\mathbf{G}}}^{(j,1,\alpha,\theta,\phi,\tau)}(z)\underline{\mathbf{A}}(z) \tag{5.25}$$

where as a result of this transformation $a'_{j1}(0) = 0$. Furthermore, $|a'_{11}(0)|^2 = |a_{j1}(\tau)|^2 + |a_{11}(0)|^2$ and $a'_{11}(0)$ is real.

This two-stage iterative process is repeated, replacing $\underline{\mathbf{A}}(z)$ with $\underline{\mathbf{A}}'(z)$ until all coefficients beneath the diagonal of the first column of the polynomial matrix $\underline{\mathbf{A}}'(z)$ are sufficiently small. As with the PQRD-BS algorithm, it is not always feasible to zero all coefficients beneath the diagonal of $\underline{\mathbf{A}}(z)$ and so instead only an approximately upper triangular polynomial matrix is required. Therefore, the iterative routine is repeated until all coefficients beneath the diagonal in the first column satisfy

$$\left|a'_{i1}(\tau)\right| < \epsilon, \tag{5.26}$$

$\forall \tau \in \mathbb{Z}$ and $i = 2, \ldots, p$, where $\epsilon > 0$ is some pre-specified small value. This completes the first step of the PQRD-BC algorithm. To begin the subsequent step, $\underline{\mathbf{A}}(z)$ is replaced

with $\underline{\mathbf{A}}'(z)$ and the process is repeated moving to the second column in the matrix. Once all coefficients of the polynomial elements beneath the diagonal of the column are sufficiently small, the algorithm moves to the next column, positioned to the right of the column it has just completed (provided it contains polynomial elements beneath the diagonal) and repeats the process driving all coefficients beneath the diagonal to zero until a suitable stopping condition similar to that of (5.26) is satisfied. The algorithm continues this process until all columns with polynomial elements beneath the diagonal have been been visited. Once all steps have been completed, this defines one sweep of the PQRD-BC algorithm. Again, as with the PQRD-BS algorithm, multiple sweeps of this algorithm can be implemented if required.

Assuming k steps of the algorithm, the polynomial matrix decomposition performed is of the form

$$\underline{\mathbf{A}}_k(z) = \underline{\mathbf{Q}}_k(z)\underline{\mathbf{A}}(z) \tag{5.27}$$

where $\underline{\mathbf{Q}}_k(z) \in \underline{\mathbb{C}}^{p\times q}$ is a paraunitary polynomial matrix, which is calculated as a series of EPGR matrices, and $\underline{\mathbf{A}}_k(z)$ is the transformed polynomial matrix, which is guaranteed to converge to an upper triangular matrix. Note that the number of steps in one sweep of the algorithm can be determined in advance, but the number of sweeps in the decomposition and the number of EPGRs required to drive all coefficients sufficiently small in each sweep cannot. The proof of convergence for the algorithm is detailed in Section 5.5.2 and a concise description of the algorithm is contained in the Appendix B.

If both the PQRD-BS and PQRD-BC algorithms are applied to the same scalar matrix, they will generate the same unitary and upper triangular matrices. Furthermore, the two algorithms will also require the same number of Givens rotations to obtain an exactly upper triangular matrix and this will be obtained, in both cases, from only one sweep of the algorithm. The number of Givens rotations required can also be specified in advance. However, computationally the PQRD-BC algorithm will be slightly more expensive than the PQRD-BS algorithm, due to the computations required to locate the dominant coefficient within a given column of the matrix. This is obviously not required within the PQRD-BS algorithm. Furthermore, when calculating the QRD of a polynomial matrix using either of these PQRD

algorithms, the number of EPGRs required to transform the polynomial matrix into an upper triangular polynomial matrix cannot be determined in advance and an approximately upper triangular matrix is only ever formulated from the decomposition. It might be expected that the PQRD-BC algorithm will require fewer sweeps and consequently less overall iterations of the algorithm, due to the stopping criterion considering all elements beneath the diagonal of a given column of the matrix at each step. For this reason, the PQRD-BC algorithm is a sensible progression from the previous PQRD-BS algorithm as it will always require fewer steps per sweep.

5.5.2 Convergence of the PQRD by Columns Algorithm

Convergence of each step of the algorithm can easily be deduced in a similar way to the proof of convergence given for the PQRD-BS algorithm. At each step, to drive the coefficients beneath the diagonal of column k sufficiently small, the quantity $E_{kk}(0)$ will increase at each iteration by the magnitude squared of the dominant coefficient. This quantity will be bounded from above by the squared Frobenius norm of the k^{th} column and so convergence can easily be proved for each step. Note that once the k^{th} column has had all of its coefficients beneath the diagonal driven sufficiently small, the diagonal coefficients from this column will not be rotated in any future steps and so the quantity $E_{kk}(0)$ will then remain constant throughout future steps of that particular sweep of the algorithm.

A proof of convergence for the algorithm does not simply follow from the proof of convergence for each step. Once again, not driving all coefficients associated with polynomial elements beneath the diagonal to be zero, can lead to small coefficients being rotated with each other and possibly increasing in magnitude. If these coefficients do increase above the threshold value of ϵ, then a second sweep of the algorithm is required. The proof of convergence now follows directly from the proof for the PQRD-BS algorithm.

The quantity $E_{11}(0)$ can only ever increase in future sweeps and so a point will be reached where no further rotations are required within the first column. Once this point has been reached, the quantity $E_{22}(0)$ will increase monotonically through all future steps until a point is reached where this quantity can no longer increase. As a result no further rotations are

required in this column. This will continue to happen to all diagonal elements situated on the coefficient plane of order zero working from left to right through the matrix.

5.6 Algorithm 3: Sequential Best Rotation PQRD

The motivation behind the third and final PQRD algorithm was to use an entirely Sequential Best Rotation (SBR) approach, as used within the PEVD algorithm SBR2. As a result, this algorithm will not need to operate as a series of steps, each requiring their own convergence criterion to be met before the algorithm can continue, and multiple sweeps of the algorithm will never be required.

The QRD of a scalar matrix could also be obtained using this algorithm, although for this case it is preferable to use an ordered technique such as the PQRD-BS algorithm. Convergence of the SBR approach is guaranteed, but this algorithm will typically require more Givens rotations to obtain an upper triangular matrix, due to the possibility of the series of rotations forcing elements that have previously been driven to zero to increase in magnitude.

5.6.1 The PQRD Algorithm by Sequential Best Rotation

The PQRD-SBR algorithm operates as an iterative process, where at each iteration the polynomial coefficient beneath the diagonal with maximum magnitude, termed the dominant coefficient, is driven to zero by applying an appropriate EPGR. Once the dominant coefficient has been driven to zero, an inverse time shift matrix is applied to the resulting matrix to ensure that the dominant coefficient is returned to its original position in the polynomial element. As a consequence, the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ is not simply generated as a series of EPGRs, as for the previous two decomposition algorithms, but instead formulated as a series of EPGRs interspersed with inverse time shift matrices. The motivation behind this step, is explained within the description of the iterative process of the algorithm. The three-step routine carried out at each iteration, to find the dominant coefficient and drive it to zero is now discussed in detail.

The algorithm begins by locating the dominant polynomial coefficient situated beneath

the diagonal of the polynomial matrix $\underline{\mathbf{A}}(z)$, i.e. it finds the coefficient $a_{jk}(t)$, for $j > k$, such that

$$|a_{jk}(t)| \geq |a_{mn}(\tau)| \tag{5.28}$$

holds for all coefficients $a_{mn}(\tau)$, where $m = 2, \ldots, p$, $n = 1, \ldots, m-1$ and for all values of the lag parameter $\tau \in \mathbb{Z}$. Unlike the previous two algorithms, if the dominant coefficient is not unique then it is better to choose the coefficient associated with the uppermost left polynomial element of the matrix as this will achieve a faster rate of convergence.

Subsequently, the dominant coefficient now needs to be driven to zero; this is achieved by applying an appropriate EPGR to the polynomial matrix $\underline{\mathbf{A}}(z)$ as follows

$$\underline{\mathbf{A}}'(z) = \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z) \tag{5.29}$$

where the indices j, k and t define the position of the dominant polynomial coefficient, which is to be eliminated. The structure of this matrix, and formulae for calculating the necessary rotation angles, is the same as the previous two algorithms and is described in detail in Section 5.2. Following the application of the EPGR the dominant coefficient has been driven to zero and so $a'_{jk}(0) = 0$, the diagonal coefficient $a'_{kk}(0)$ is real and will have increased in magnitude squared such that $|a'_{kk}(0)|^2 = |a_{kk}(0)|^2 + |a_{jk}(0)|^2$.

An elementary inverse time-shift matrix $\underline{\mathbf{B}}^{(j,t)}(z)$ takes the form of an identity matrix with the exception of the j^{th} diagonal element which is z^{-t}, i.e.

$$\underline{\mathbf{B}}^{(j,t)}(z) = \begin{bmatrix} \mathbf{I}_{j-1} & 0 & 0 \\ 0 & z^{-t} & 0 \\ 0 & 0 & \mathbf{I}_{p-j} \end{bmatrix}. \tag{5.30}$$

This matrix is applied to $\underline{\mathbf{A}}'(z)$ to generate the transformed matrix

$$\underline{\mathbf{A}}''(z) = \underline{\mathbf{B}}^{(j,t)}(z)\underline{\mathbf{A}}'(z), \tag{5.31}$$

where consequently all polynomial elements on the j^{th} row of $\underline{\mathbf{A}}'(z)$ have had a t-fold delay

applied to them. In particular this is done to move the diagonal coefficient associated with the lag t back to its original position on the coefficient plane of order zero, i.e. $a''_{jj}(0) = a'_{jj}(-t)$. More specifically, following the application of the matrix, the elements on the j^{th} row of the matrix are shifted accordingly

$$a''_{jm}(\tau) = a'_{jm}(\tau - t) \quad \text{for} \quad m = 1, ..., q \quad \text{and} \quad \forall \tau \in \mathbb{Z}. \tag{5.32}$$

All other rows of the polynomial matrix are not affected by this transformation. Note that the matrix demonstrated by equation (5.30) is the inverse of the time-shift matrix that is incorporated in the EPGR applied in the second step of the iterative process. This final step is implemented to ensure that the original zero-lag diagonal elements are brought back onto the zero-lag plane following the application of every EPGR, to minimise the disruptive effects of the initial time shifts incorporated in the EPGR. At each iteration the zero lag diagonal coefficient in the same column of the matrix as the dominant coefficient, for this example the coefficient $a_{kk}(0)$, will increase in magnitude squared. For convergence of the algorithm, it is better to keep these diagonal coefficients positioned on the coefficient matrix of order zero, as much as possible. However, by applying the elementary matrix delay incorporated in the EPGR the coefficient $a_{jj}(0)$ will be shifted and so this is not possible. In the SBR2 algorithm, this is not an issue, as the elementary delay matrices are applied on both sides of the polynomial matrix and so the set of zero lag diagonal coefficients will never be affected by the delay matrices. This cannot be done here, where it is required that the paraunitary transformation matrices are only applied on the left hand-side of the polynomial matrix. Note also that this point is not an issue with the two previous PQRD algorithms as convergence is guaranteed by using two ordered approaches.

At the end of the first iteration of the PQRD-SBR algorithm the following $p \times p$ paraunitary matrix has been calculated

$$\underline{\mathbf{Q}}_1(z) = \underline{\mathbf{B}}^{(j,t)}(z)\underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z). \tag{5.33}$$

The three-stage routine is now repeated replacing $\underline{\mathbf{A}}(z)$ with $\underline{\mathbf{A}}''(z)$, until the magnitudes of

all coefficients situated beneath the diagonal are sufficiently small and the following stopping condition is satisfied

$$|a_{jk}(\tau)| < \epsilon \tag{5.34}$$

$\forall \tau \in \mathbb{Z}$, for $j = 1, \ldots, p$, $k = 1, \ldots, q$ where $j > k$ and $\epsilon > 0$ is a pre-specified small value.

Following i iterations of the algorithm, the matrix decomposition performed can be expressed as

$$\underline{\mathbf{Q}}_i(z) \ldots \underline{\mathbf{Q}}_2(z)\underline{\mathbf{Q}}_1(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{A}}_i(z), \tag{5.35}$$

where $\underline{\mathbf{Q}}_i(z) \in \underline{\mathbb{C}}^{p \times p}$ is a paraunitary polynomial matrix of the form $\underline{\mathbf{Q}}_i(z) = \underline{\mathbf{B}}^{(j,t)}(z)\underline{\mathbf{G}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ and $\underline{\mathbf{A}}_i(z)$ is the transformed polynomial matrix resulting from i iterations of the algorithm. The algorithm is guaranteed to converge and so this matrix will converge to an upper triangular matrix.

Once the algorithm has converged, a diagonal matrix of final phase adjustment terms can be applied to resulting upper triangular polynomial matrix, to ensure that the diagonal zero lag coefficients are all real and positive. This will then ensure uniqueness of the decomposition in the scalar matrix case, provided the matrix is non-singular. The same must also be done to the coefficient associated with the zero lag of the final diagonal element of the upper triangular matrices obtained from the first two algorithms, for the same reason.

5.6.2 Convergence of the PQRD Algorithm by Sequential Best Rotation

This algorithm operates as a series of iterations, each one designed to zero the dominant coefficient from anywhere beneath the diagonal of the matrix. At each iteration of the algorithm, to zero the dominant coefficient $a_{jk}(t)$, the quantity $|a''_{kk}(0)|^2$ will increase by the magnitude squared of the dominant coefficient, i.e. $|a''_{kk}(0)|^2 = |a'_{kk}(0)|^2 = |a_{kk}(0)|^2 + |a_{jk}(t)|^2$. However, at the same time it is possible that energy can move between any other coefficients on the k^{th} row and j^{th} row of the matrix over all possible lags. Note that the quantity $|a''_{kk}(0)|^2$ is not guaranteed to increase monotonically over subsequent iterations if it involves rotating

coefficients in the same row (row k) positioned to the left of column k, i.e. EPGRs applied to any coefficients associated with polynomial elements $\underline{a}_{k1}(z), \ldots, \underline{a}_{k(k-1)}(z)$ can force the quantity $|a''_{kk}(0)|^2$ to decrease. In fact, there are three possible ways in which this quantity can be affected,

1. This quantity will increase if the dominant coefficient is beneath the diagonal of the k^{th} column of the matrix.

2. The quantity will be unaffected by any rotations to zero a dominant coefficient positioned in a column to the right of the k^{th} column, i.e. if the dominant coefficient is in columns $k+1, \ldots, q$. Both the EPGR and the inverse time shift will not affect any polynomial elements in the k^{th} row of the matrix and so the quantity is guaranteed to not decrease. Furthermore, if the dominant coefficient is in a column positioned to the left of column k, but in any row beneath row k, it is also unaffected.

3. If the dominant coefficient at a future iteration is positioned in row k, but in a column to the left of column k, then it is possible for this quantity to decrease as it will be affected by both the application of the EPGR and the inverse time shift matrix.

However, any EPGRs applied to polynomial coefficients in the first column of $\underline{\mathbf{A}}(z)$ will lead to an increase in the quantity $|a''_{11}(0)|^2$ and this will continue through all iterations of the algorithm. This quantity will remain unaffected over all future iterations of the algorithm, even if they involve applying rotations to polynomial coefficients that are positioned in columns to the right of the first column. Furthermore, this quantity will never be affected by the application of elementary delay matrices in either the first or third stages of each iteration of the algorithm. Therefore, over all iterations of the algorithm, this quantity will be monotonically increasing and, as the paraunitary transformations are norm preserving in the columns of the matrix, the quantity is bounded above by the squared Frobenius norm of all elements in the first column of $\underline{\mathbf{A}}(z)$. In a similar way to the previous two algorithms, this quantity will have a supremum and so there exists an iteration by which the magnitude squared of the dominant coefficient in the first column is bounded by ϵ.

As with the previous algorithms, once this has been achieved, future rotations can cause the squared Frobenius norm of elements beneath the diagonal in the first column to be redistributed, forcing coefficients to possibly increase in magnitude and become larger than ϵ. If this happens, then future EPGRs will need to be applied to coefficients in the first column. However, this process cannot continue indefinitely. There will be a point, as with the previous two algorithms, where no future rotations are required to be applied to any coefficients beneath the diagonal of the first column. Subsequently, the quantity $|a''_{22}(0)|^2$ will increase monotonically, until a point is reached where the stopping condition is satisfied and no further rotations are required in the second column of the polynomial matrix. This process continues through all columns of the matrix working from left to right.

5.7 Non-Uniqueness of Solutions

The scalar matrix QRD is unique provided the input matrix is non-singular and the diagonal elements of the resulting upper triangular matrix are positive and real. Similarly, for a non-singular polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times p}$, provided it is of full column rank and assuming two PQRDs exist for this matrix, such that

$$\underline{\mathbf{A}}(z) = \underline{\mathbf{Q}}_1(z)\underline{\mathbf{R}}_1(z) \quad \text{and} \quad \underline{\mathbf{A}}(z) = \underline{\mathbf{Q}}_2(z)\underline{\mathbf{R}}_2(z) \tag{5.36}$$

then

$$\underbrace{\widetilde{\underline{\mathbf{Q}}}_2(z)\underline{\mathbf{Q}}_1(z)}_{\underline{\mathbf{G}}(z)} = \underbrace{\underline{\mathbf{R}}_2(z)\underline{\mathbf{R}}_1^{-1}(z)}_{\underline{\mathbf{T}}(z)}. \tag{5.37}$$

The polynomial matrix $\underline{\mathbf{G}}(z)$ will be paraunitary by construction and $\underline{\mathbf{T}}(z)$ will be an upper triangular polynomial matrix as it is formulated as the product of two upper triangular polynomial matrices. Furthermore, according to equation (5.37) the polynomial matrix $\underline{\mathbf{T}}(z)$ must also be paraunitary and therefore satisfy

$$\underline{\mathbf{T}}(z)\widetilde{\underline{\mathbf{T}}}(z) = \widetilde{\underline{\mathbf{T}}}(z)\underline{\mathbf{T}}(z) = \mathbf{I}. \tag{5.38}$$

By equating both side of equation (5.38) it can be demonstrated that an upper triangular polynomial matrix can only be paraunitary if it is a diagonal matrix where the diagonal polynomial elements satisfy $\left\|\underline{t}_{jj}(z)\right\|_F^2 = 1$ for $j = 1,\ldots,p$. The diagonal element $\underline{t}_{jj}(z)$ could be calculated as

$$\begin{aligned}\underline{t}_{jj}(z) &= \left[\underline{\mathbf{R}}_2(z)\right]_{jj}\left[\underline{\mathbf{R}}_1^{-1}(z)\right]_{jj} \\ &= \frac{1}{det(\underline{\mathbf{R}}_1(z))}\left[\underline{\mathbf{R}}_2(z)\right]_{jj}\prod_{\substack{k=1\\k\neq j}}^{p}\left[\underline{\mathbf{R}}_1(z)\right]_{kk} \\ &= \frac{\left[\underline{\mathbf{R}}_2(z)\right]_{jj}}{\left[\underline{\mathbf{R}}_1(z)\right]_{jj}} \end{aligned} \tag{5.39}$$

where for uniqueness, it is required that this quantity is equal to unity for $j = 1,\ldots,p$, but as $\underline{t}_{jj}(z)$ is a polynomial for each value of j, this cannot be imposed. If the input matrix is a scalar, then uniqueness of the solutions is easily enforced by requiring that the diagonal elements of the upper triangular matrix obtained by the QRD are real and positive. However, for a polynomial matrix it is not so simple and the only thing known about the diagonal elements of the polynomial matrix $\underline{\mathbf{T}}(z)$ is that $\left\|\underline{t}_{jj}(z)\right\|_F^2 = 1$ for $j - 1,\ldots,p$. Therefore, it is possible to have a diagonal paraunitary matrix $\underline{\mathbf{T}}(z)$, whose diagonal elements will consist of time-shift and phase adjustment terms of the form $t_{jj}(\tau) = e^{i\alpha}z^{-\tau}$ for $j = 1,\ldots,p$ and all time lags τ, such that $\underline{\mathbf{T}}(z)\underline{\mathbf{R}}_1(z) = \underline{\mathbf{R}}_2(z)$, where both $\underline{\mathbf{R}}_1(z)$ and $\underline{\mathbf{R}}_2(z)$ are both upper triangular. The same is true of the paraunitary matrix obtained from the PQRD. Note that for the application of the PQRD to MIMO communication problems, uniqueness of the solutions is not required and therefore is of no direct relevance.

5.7.1 Implementation of the PQRD Algorithms

The truncation method suitable for non para-Hermitian polynomial matrices detailed in Section 4.3.2, is used within the PQRD algorithm to ensure that the order of the polynomial matrix $\underline{\mathbf{A}}(z)$ does not grow unnecessarily large. The same truncation method can also be applied to the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$, as the order of this matrix can also become unnecessarily large.

As with the SBR2 algorithm, it is not necessary to apply the polynomial Givens rotation to the entire polynomial matrix $\underline{\mathbf{A}}(z)$ at each iteration of the algorithm. Only two rows of the matrix are affected at each iteration and so only these two rows need to be updated. Similarly, the transformation matrix $\underline{\mathbf{Q}}(z)$ is not computed within the iterative routine of the algorithm. Instead the parameters j, k, θ, α, ϕ and t are stored and so the resulting transformation matrix can be calculated afterwards, when required. As a result, this will often help reduce the computational load of the algorithm enabling whichever of the PQRD algorithms has been used, to run faster.

5.8 Numerical Example

The objective of this example is to illustrate the results of the three different PQRD algorithms when each is applied to a fairly simple polynomial matrix of a relatively small size and order. A polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{4\times 3}$ of order 4 was generated, where the real and imaginary parts of the coefficients of each of the polynomial elements were randomly drawn from a normal distribution with mean zero and unit variance. A graphical representation for this polynomial matrix can be seen in Figure 5.2, where a stem plot is used to demonstrate the magnitude of the series of coefficients for each of the polynomial elements. The position of the stem plot in the figure relates to the position of the polynomial element, which it represents within the matrix.

The three PQRD algorithms were applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ in turn, each time using the energy based truncation method suitable for non para-Hermitian polynomial matrices with the truncation parameter set as $\mu = 10^{-6}$. The stopping condition for each of the three PQRD algorithms was appropriately set to ensure that the magnitude of each polynomial coefficient beneath the diagonal of the resulting polynomial matrix $\underline{\mathbf{R}}(z)$ was less than 10^{-3}, allowing multiple sweeps of the PQRD-BS and PQRD-BC algorithms to be implemented if required. The upper triangular and paraunitary polynomial matrices obtained when the PQRD-BS algorithm was applied to $\underline{\mathbf{A}}(z)$ are given in Figures 5.3 and 5.4 respectively. Figures 5.5 and 5.6 illustrate the matrices obtained from the PQRD-BC algorithm

and Figures 5.7 and 5.8 the results from the PQRD-SBR algorithm. It is clearly visible from these figures that neither the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ nor the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ obtained from any of three decomposition algorithms is unique, although similarities between the matrices can be seen.

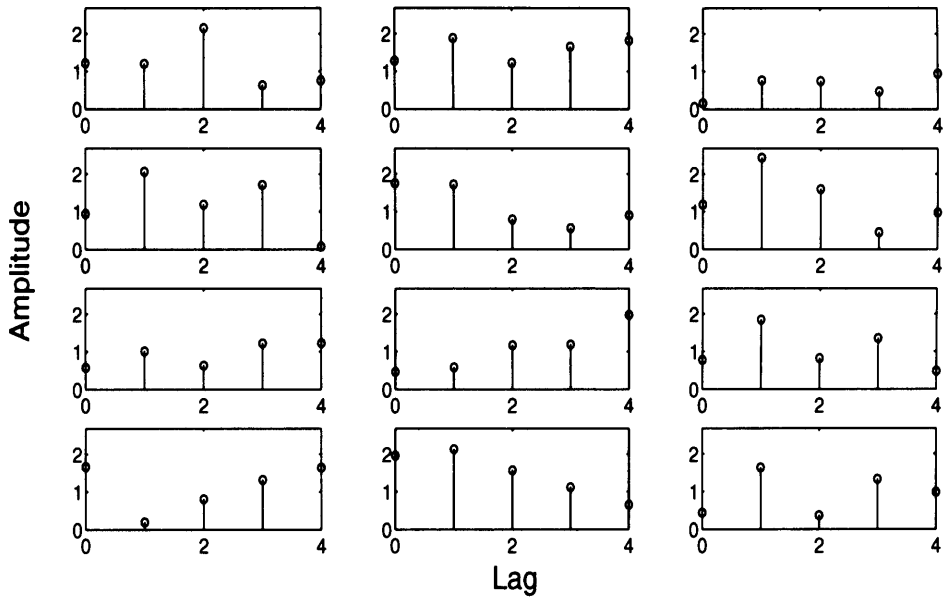

Figure 5.2: A stem plot representation of the series of coefficients of the polynomial matrix $\underline{\mathbf{A}}(z)$, to be used as input to each of the three algorithms for calculating the PQRD.

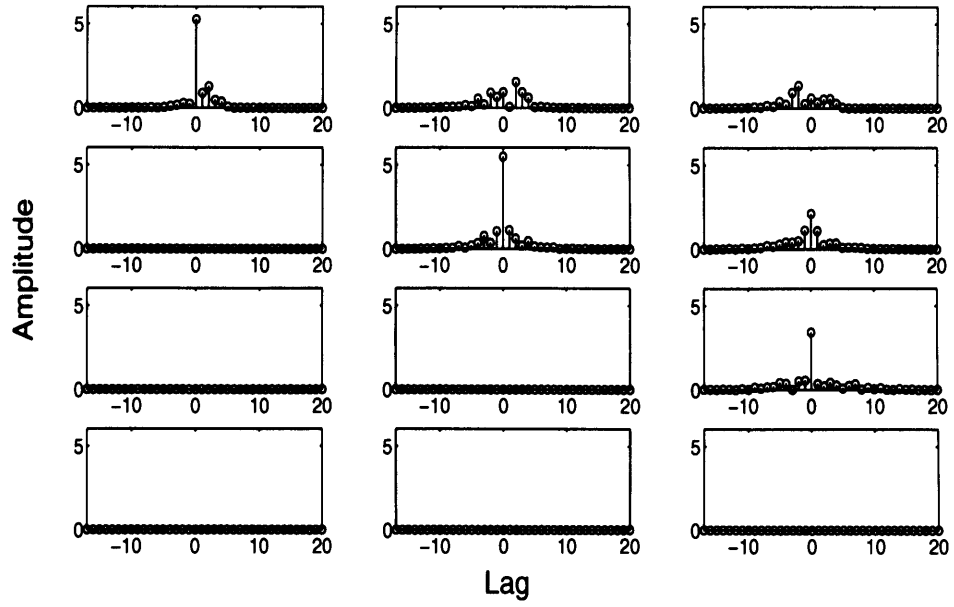

Figure 5.3: The upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BS algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}(z)$.

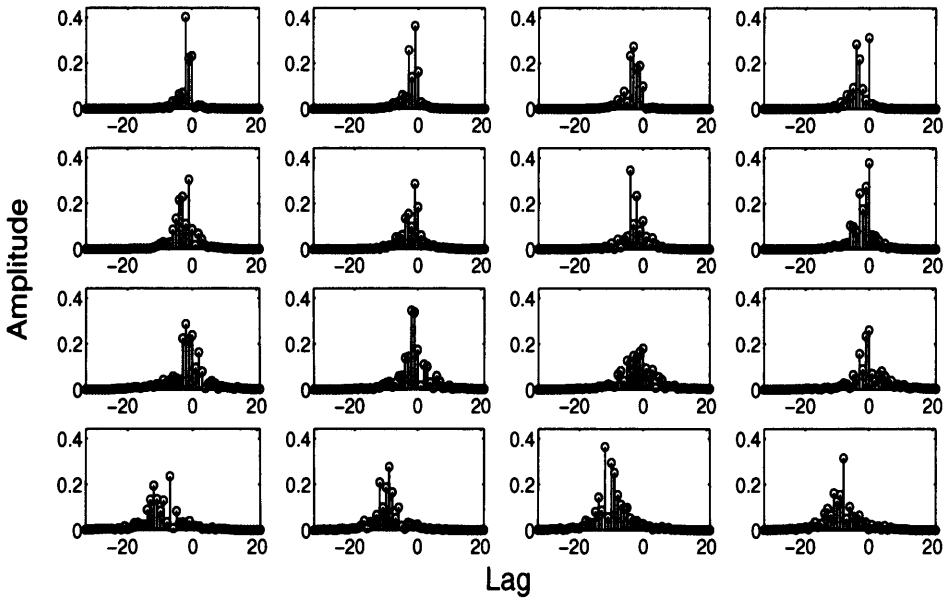

Figure 5.4: The paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BS algorithm with polynomial input matrix $\underline{\mathbf{A}}(z)$.

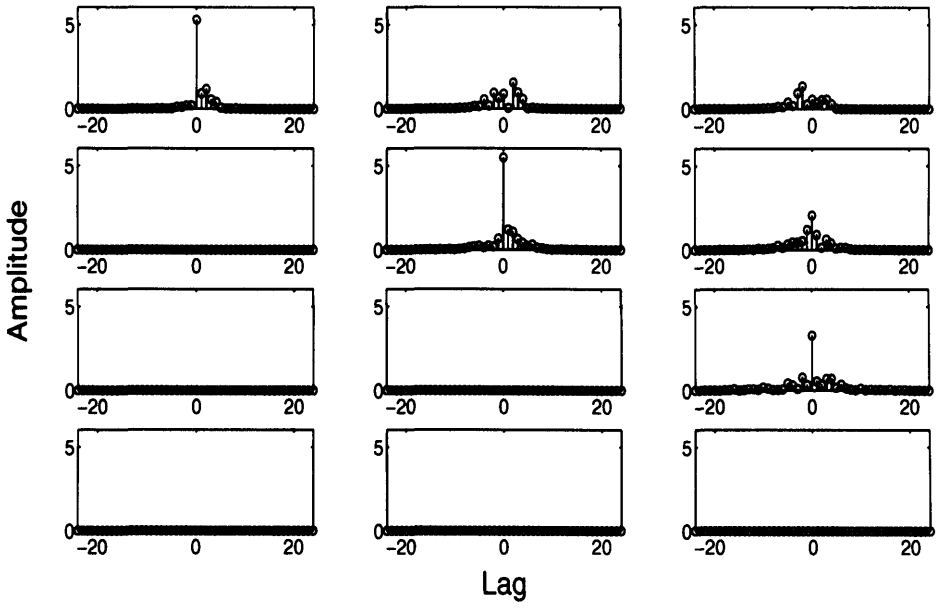

Figure 5.5: The upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BC algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}(z)$.

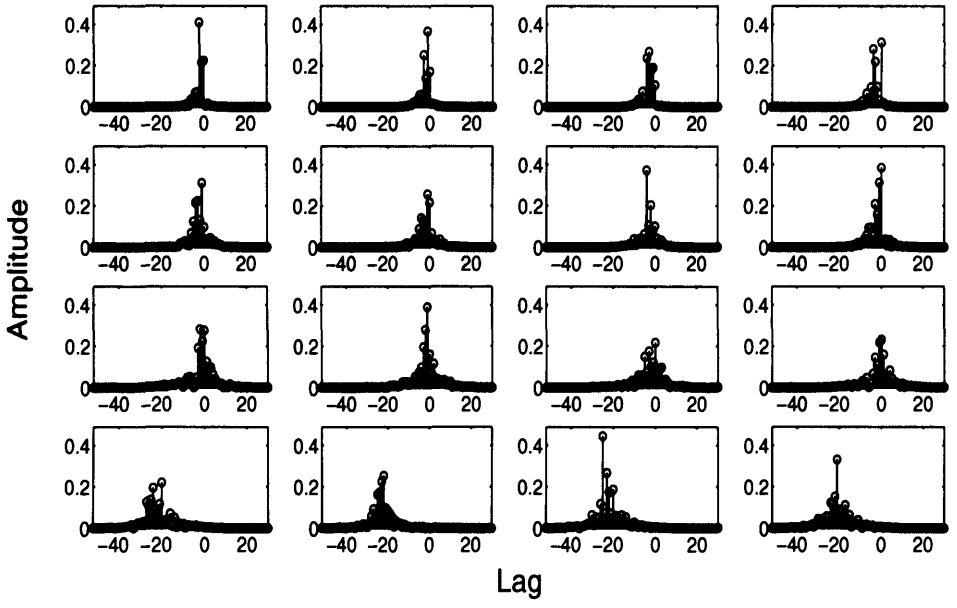

Figure 5.6: The paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BC algorithm with input matrix $\underline{\mathbf{A}}(z)$.

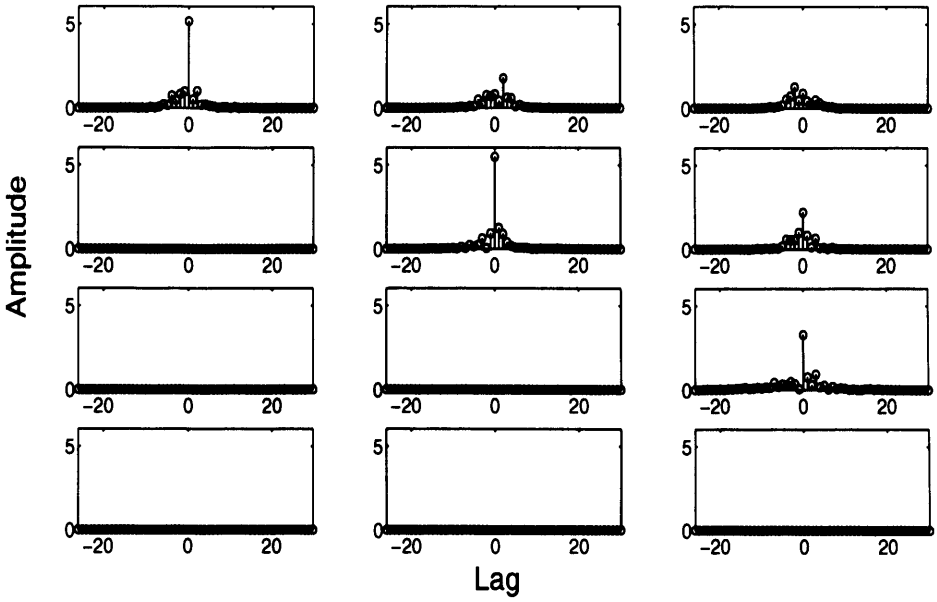

Figure 5.7: The upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-SBR algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}(z)$.

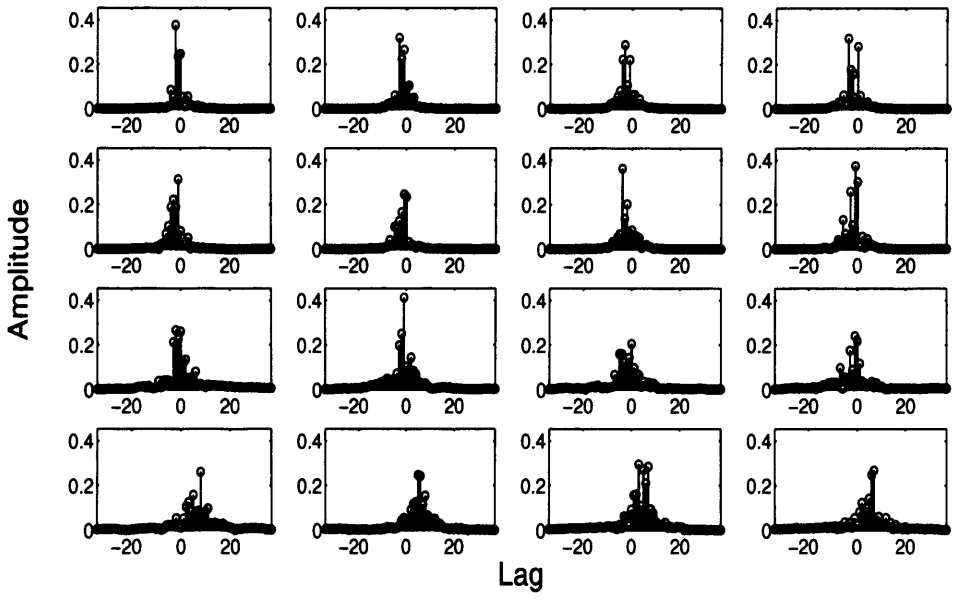

Figure 5.8: The paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-SBR algorithm with input matrix $\underline{\mathbf{A}}(z)$.

The relative error for the decomposition performed is defined as

$$E_{rel} = \frac{\left\| \underline{\mathbf{A}}(z) - \underline{\tilde{\mathbf{Q}}}(z)\underline{\mathbf{R}}'(z) \right\|_F}{\|\underline{\mathbf{A}}(z)\|_F}, \tag{5.40}$$

where $\underline{\mathbf{R}}'(z)$ is equal to the approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ with all coefficients beneath the diagonal set to zero. This measure is calculated to ensure that the accuracy of the decomposition obtained is not compromised by either using the truncation method or by setting all non-zero coefficients beneath the diagonal to zero, as required for the application of the PQRD to MIMO communications, discussed further in Chapter 8. The results of the three decompositions are presented in Table 5.1, where the measure L defines the Frobenius norm of the elements beneath the diagonal of the transformed polynomial matrix following each iteration, i.e. the quantity

$$L = \sqrt{\sum_{\tau} \sum_{j=2}^{p} \sum_{k=1}^{j-1} \left| a'_{jk}(\tau) \right|^2} \tag{5.41}$$

where $\underline{\mathbf{A}}'(z)$ is the transformed polynomial matrix following the application of an EPGR and p defines the number of rows of the polynomial matrix $\underline{\mathbf{A}}'(z)$ and, for this example, is equal to four.

	PQRD-BS	**PQRD-BC**	**PQRD-SBR**
Number of EPGRs	547	556	1315
Number of Sweeps	1	1	-
Final Order of $\underline{\mathbf{R}}(z)$	37	48	56
Final Order of $\underline{\mathbf{Q}}(z)$	54	81	71
E_{rel}	8.44×10^{-3}	8.66×10^{-3}	1.34×10^{-2}
Final value of g	9.99×10^{-4}	9.79×10^{-4}	9.99×10^{-4}
Final value of L	5.60×10^{-3}	5.33×10^{-3}	9.30×10^{-3}
Computational Time (Seconds)[1]	0. 52	0.65	4.13

Table 5.1: Results obtained from applying the three algorithms for computing the PQRD to $\underline{\mathbf{A}}(z)$.

From Table 5.1, it can be seen that the PQRD-BS algorithm converged in the fewest number of EPGRs over all steps of the algorithm, although the performance was similar to the PQRD-BC algorithm. The PQRD-SBR algorithm required over double the number of EPGRs compared to the other two algorithms and therefore took considerably more time to calculate. Note that both the PQRD-BS and PQRD-BC algorithms required only a single sweep to converge.

Figure 5.9 illustrates the quantity L demonstrated by equation (5.41) over the complete series of iterations of each of the three PQRD algorithms. If the decomposition algorithm uses a process of steps, such as the PQRD-BS and PQRD-BC algorithms, this is easily visible from this figure. Initially, the PQRD-SBR algorithm appears to perform considerably better than the other two algorithms, as at each iteration the largest coefficient from anywhere beneath the diagonal of the matrix will be driven to zero. This will not happen with either the PQRD-BS or PQRD-BC algorithms, unless the largest coefficient is positioned in either the polynomial element $\underline{a}_{21}(z)$ (if using the PQRD-BS algorithm) or is situated beneath the diagonal of the first column of the matrix (if using the PQRD-BC algorithm). However, although the PQRD-SBR algorithm does not require any steps or sweeps, it takes far more EPGRs to reach a similar level of convergence as the other two PQRD algorithms. This is due to the proof of convergence for each of the algorithms, operating through the columns of the matrix from left to right. An algorithm that operates using a series of steps, working through the matrix from left to right, will typically converge quicker.

Finally, Figure 5.10 demonstrates the same measure L, this time for the PQRD-SBR algorithm with and without applying the inverse time-shift matrix in the third step of each iteration of the algorithm. At each iteration, for example, to zero the polynomial coefficient $a_{jk}(\tau)$ there are three possible ways the quantity $|a_{kk}(0)|^2$ can be affected. These are listed in the proof of convergence of the PQRD-SBR algorithm. However, by applying the appropriate EPGR to zero $a_{jk}(\tau)$, the coefficient $a_{jj}(0)$ will have a delay of size τ applied to it and so will no longer be positioned on the coefficient matrix of order zero. If the inverse delay step is not applied to move it back to its original position following the rotation, then at a subsequent

[1]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

iteration to zero a coefficient associated with elements situated beneath the diagonal of the j^{th} column of the matrix, a different coefficient will be increasing in magnitude squared and it is not guaranteed that this coefficient will continue to increase (in magnitude squared) over future iterations to zero coefficients beneath the diagonal of the j^{th} column. Allowing the zero-lag diagonal coefficients to move over a series of iterations, will mean that the algorithm is not consistently forcing the same coefficients on the zero-lag coefficient matrix to increase in magnitude squared by application of each EPGR. Therefore the PQRD-SBR algorithm when not using the inverse time-shift will result in erratic behaviour as demonstrated by Figure 5.10, clearly demonstrating why the inverse time shift is required within the PQRD-SBR algorithm.

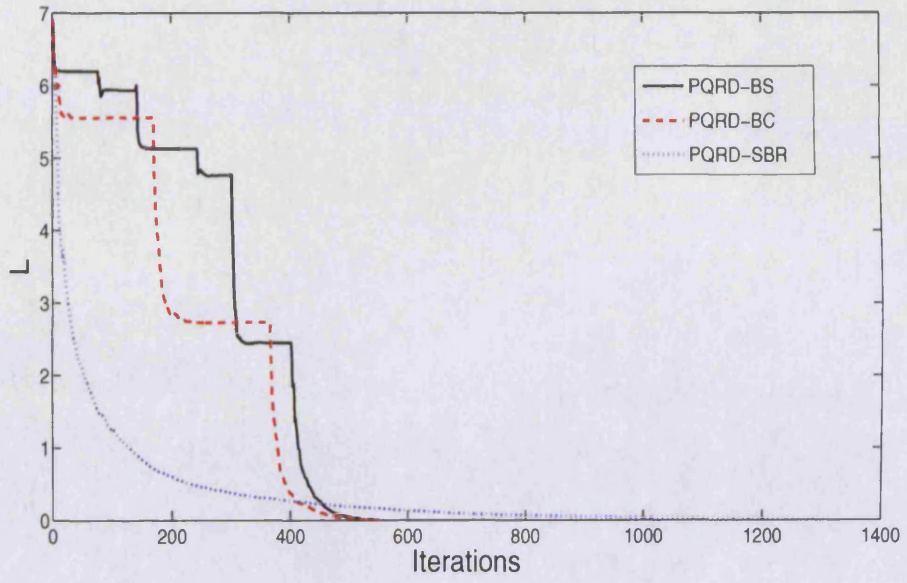

Figure 5.9: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix at each iteration of each of the PQRD algorithms.

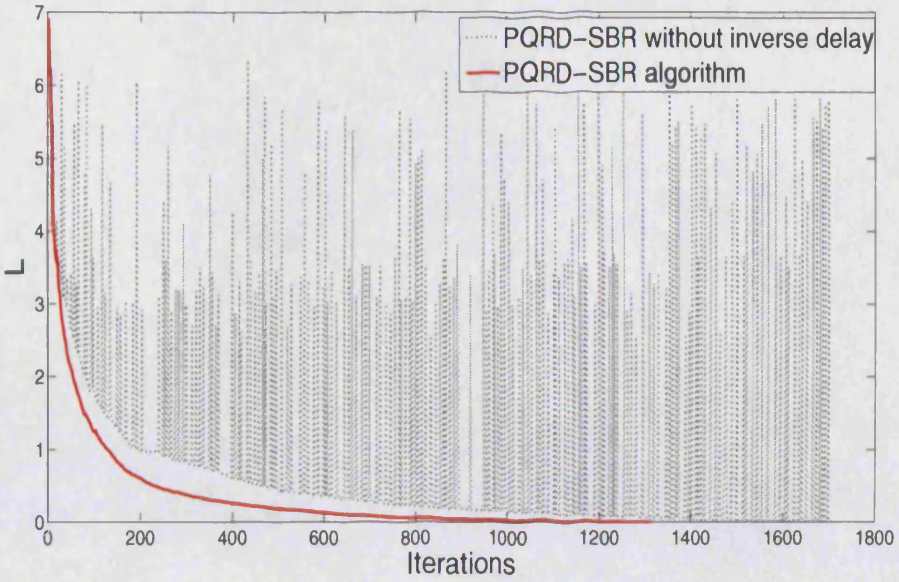

Figure 5.10: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix at each iteration when applying the PQRD-SBR algorithm with and without using the inverse delay step.

Finally, Figure 5.11 demonstrates the polynomial matrix obtained from calculating the inverse decomposition $\underline{\widehat{\mathbf{A}}}(z) = \underline{\widetilde{\mathbf{Q}}}(z)\underline{\mathbf{R}}(z)$ and compares this matrix to the input matrix. For this example, the polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$ were found using the PQRD-BS algorithm. Note that only the five lags $\underline{\widehat{\mathbf{A}}}(0), \ldots, \underline{\widehat{\mathbf{A}}}(4)$ are demonstrated in this figure. By inspection of this figure, it can be observed that a very accurate polynomial matrix decomposition can be obtained using the PQRD-BS algorithm. Note that the coefficients matrices not included in this figure, i.e. those outside the five lags $\underline{\widehat{\mathbf{A}}}(0), \ldots, \underline{\widehat{\mathbf{A}}}(4)$, account for 0.0071% of $\left\|\underline{\widehat{\mathbf{A}}}(z)\right\|_F^2$.

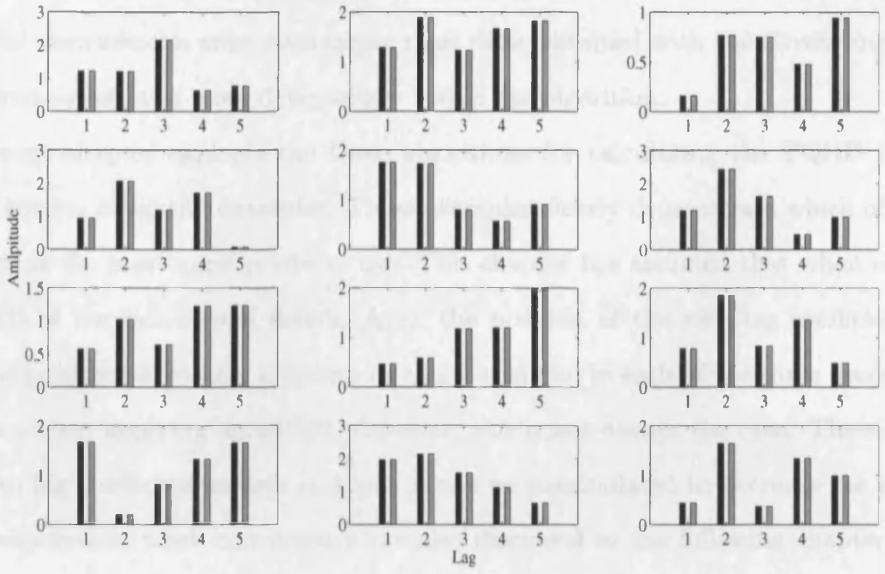

Figure 5.11: A stem plot representation of the series of coefficients of the polynomial input matrix $\underline{\mathbf{A}}(z)$ in green, and the matrix obtained from calculating the inverse decomposition $\underline{\tilde{\mathbf{Q}}}(z)\underline{\mathbf{R}}(z)$ when the decomposition was formulated using the PQRD-BS algorithm demonstrated in blue.

5.9 Conclusions

This chapter has introduced three algorithms for calculating the QR decomposition of a polynomial matrix. Each of the algorithms can be applied to a polynomial matrix of any size and order, where the coefficients of the elements can be either real or complex, and has been proven to converge. The computational complexity of each of the algorithms is formulated in Appendix C. The polynomial matrices obtained from the PQRD are not unique, however, this is not a problem for the potential application of the decomposition discussed in Chapter 8. If any of the algorithms are applied to a non-singular matrix of scalars, then uniqueness can be guaranteed by ensuring that the diagonal elements of the upper triangular matrix are positive and real. However, this cannot be done with the PQRD, where each diagonal element of this matrix is a polynomial with an associated set of coefficients. Each of the three algorithms operate using Givens rotations interspersed with elementary delay matrices. Appendix A contains an example where Householder reflections have been adapted to enable

the calculation of the QRD of a polynomial matrix. With this method, the orders of the polynomial matrices can grow even larger than those obtained with the Givens method, due the requirement of even more delay stages within the algorithm.

The next chapter explores the three algorithms for calculating the PQRD further by detailing several insightful examples. These examples clearly demonstrate which of the three algorithms is the most appropriate to use. This chapter has assumed that when calculating the PQRD of the polynomial matrix $\underline{\mathbf{A}}(z)$, the position of the zero lag coefficient matrix within the polynomial matrix is known as this is required in each of the three decomposition algorithms when applying an EPGR. However, this is not always the case. The significance of the zero lag coefficient matrix and how it can be manipulated to decrease the number of EPGRs required to reach convergence are also discussed in the following chapter. Finally, the potential application of the PQRD to MIMO communications is discussed in Chapter 8 of this thesis.

Chapter 6

Discussion and Examples of the Algorithms for Calculating the QR Decomposition of a Polynomial Matrix

The objective of this chapter is to illustrate how each of the three algorithms for calculating the QRD of a polynomial matrix, which were previously introduced in Chapter 5, operate through several insightful examples. In particular, the set of examples demonstrate that the PQRD-BC algorithm is generally the best algorithm to use, as it typically requires the fewest number of EPGRs to converge and therefore requires the least amount of computational time. Note that this is not the case if the three algorithms are applied to a scalar matrix. For this case, the PQRD-BS algorithm will be computationally the least expensive, as it will not require any search routine to locate the dominant element at each iteration. However, with polynomial matrices there is an added dimension to the problem as each element of the matrix is a polynomial with an associated set of coefficients, which must all be driven sufficiently small in magnitude to achieve a PQRD. Furthermore, the order of the polynomial matrix grows at each iteration of each of the algorithms and so the best choice of algorithm for calculating the QRD of a scalar matrix, is not necessarily the best technique of achieving the same decomposition of a polynomial matrix. Note that throughout this thesis an iteration of any of the PQRD algorithms, will refer to the process taken zero a single coefficient of the polynomial matrix. In the PQRD-BS and PQRD-BC algorithms this will involve a single

EPGR. However, with the PQRD-SBR algorithm an iteration will consist of an EPGR and an inverse time-shift matrix.

For some polynomial matrices, the performance of the PQRD-BS algorithm may be comparable to the PQRD-BC algorithm, as demonstrated by the example given in the previous chapter, but this is not consistently the case. Furthermore, the set of examples presented in this chapter demonstrates the PQRD-SBR algorithm to have the worst performance of the three algorithms, as it typically requires significantly more iterations to converge, where each iteration of this algorithm is generally computationally more expensive than a single iteration of either the PQRD-BS or PQRD-BC algorithms.

Before detailing the set of numerical examples, this chapter firstly discusses the significance of the coefficient matrix containing all coefficients associated with the zero-lag, i.e. the set of coefficients of z^0, of each of the polynomial elements of the matrix to be decomposed. For the numerical example in Chapter 5, the location of this coefficient matrix within the polynomial input matrix $\underline{\mathbf{A}}(z)$ was specified as an input parameter to the algorithm. However, as the zero-lag coefficient matrix is a fundamental element of the proofs of convergence for each of the algorithms[1], the set of examples aims to determine if the position of this matrix can be chosen to reduce the number of iterations required for each algorithm to converge. In particular, a specification step for the zero-lag coefficient matrix is introduced; this can be implemented within all three of the PQRD algorithms at the outset of each iteration. This additional step generally increases the computational load of the algorithm and is therefore only an advantage in any of the three algorithms, if it significantly reduces the number of iterations required to reach convergence. The set of worked examples in this chapter indicate that this additional step can be effective at reducing the computational time of the PQRD-SBR algorithm by considerably reducing the number of iterations required for the algorithm to converge. However, this additional step does not guarantee improved performance of all of the PQRD algorithms and this is demonstrated by the set of examples, particularly when it is used within the PQRD-BS and PQRD-BC algorithms.

[1]For more information refer to the individual proofs of convergence for each of the PQRD algorithms detailed in Chapter 5.

6.1 Importance of the Zero-Lag Coefficient Matrix

The zero-lag coefficient matrix of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ is defined to be the matrix containing the coefficients of z^0 from each of the polynomial elements, i.e. the coefficient matrix $\mathbf{A}(0) \in \mathbb{C}^{p\times q}$. Alternatively, this matrix can also be referred to as the coefficient matrix of order zero. This matrix has been mentioned numerous times in the previous chapter, most significantly when discussing the process implemented within all three algorithms to drive a dominant coefficient to zero by applying an EPGR. Incorporated within this EPGR matrix is an elementary delay matrix, which is applied to the polynomial matrix with the objective of moving the dominant coefficient so that it becomes the coefficient of z^0, before applying the appropriate Givens rotation. Currently, in each of the algorithms for calculating the PQRD, the user must specify the position of the zero-lag coefficient matrix as an input parameter to the algorithm, as was done for the numerical example of Chapter 5.

Alternatively, the EPGR could have been designed to always move the dominant coefficient onto a different coefficient matrix of the polynomial matrix. However, whichever coefficient matrix is chosen it is important for the convergence of whichever of the three algorithms has been used, to keep the choice consistent throughout all iterations. The zero-lag coefficient matrix is the most appropriate choice, so that if the algorithm is applied to a matrix with scalar elements then it will generate an upper triangular matrix also with scalar entries. If a different coefficient matrix is chosen, then a polynomial upper triangular matrix would be generated, which is clearly unnecessary. Furthermore, the choice of a zero-lag coefficient matrix is irrelevant for the potential application of this decomposition detailed in Chapter 8, where the polynomial channel matrix is representative of an LTI system and is therefore invariant to time-shifts. However, the choice of this coefficient matrix will affect the decomposition performed due to the non-uniqueness of the PQRD. In particular, specifying different coefficient matrices of the polynomial input matrix to be the zero-lag coefficient matrix, will affect both the number of iterations required for the algorithm to converge and also the orders of the two polynomial matrices generated by the decomposition.

6.2 Worked Examples

The comparative performance of the three algorithms introduced in Chapter 5 for calculating the PQRD is now illustrated by applying each algorithm to the same representative set of polynomial matrices. This set of examples has been chosen to include matrices of varying dimensions, some with complex and some with real polynomial element coefficients. The set of examples also includes one polynomial input matrix, which not of full generic[2] column rank.

There are two parameters that can affect the performance of the three algorithms for each example, these are the stopping criterion ϵ and the truncation parameter μ, which must both be chosen in a similar manner to the SBR2 algorithm, to optimise the speed and accuracy of the matrix decomposition performed. Note that for the potential application of the PQRD to MIMO communications, an exactly upper triangular polynomial matrix is required and so any approximations made beneath the diagonal, which are determined by the choice of ϵ, will affect the accuracy of the decomposition performed. For the set of worked examples in this chapter, the same values for these two parameters are used, which have been chosen to enable each of the algorithms to calculate a fairly accurate PQRD without resulting in polynomial matrices of unnecessarily large orders. However, if a quicker computational time is required this can easily be resolved by setting the values of ϵ and μ higher, although this will compromise the accuracy of the decomposition performed. Alternatively, if a more accurate decomposition is required, then the values of these parameters can be reduced.

Finally, the QRD of a polynomial matrix is not unique and so the upper triangular and paraunitary matrices obtained by each of the three algorithms when given the same polynomial input matrix, will not be the same. However, these examples do not aim to compare the output matrices, but to demonstrate, which algorithm typically requires the fewest EPGRs to converge to approximately the same level of accuracy in the decomposition.

[2]This means the matrix is rank deficient for all values of the indeterminate variable z.

Computational Complexity

It is difficult to compare the computational complexity of the three algorithms, as this is entirely dependent upon the order of the two polynomial matrices at each iteration and cannot be determined in advance. Furthermore, the observed orders at each iteration will be different for each of the three algorithms and so no direct comparison can be made. However, if it assumed that each algorithm is applied to the same polynomial matrix for only a single iteration, then the PQRD-BS algorithm will be computationally the least expensive of the three PQRD algorithms. All three algorithms implement the same routine to apply an EPGR, however, each differs in the search routine to locate the dominant coefficient. The PQRD-BS algorithm searches for the dominant coefficient in only one polynomial element at each iteration. The PQRD-BC algorithm searches through all polynomial elements beneath the diagonal of one column of the matrix and the PQRD-SBR algorithm through all polynomial elements beneath the diagonal of the polynomial matrix. With respect to a single iteration, each algorithm is therefore computationally more expensive than the previous. Furthermore, the PQRD-SBR algorithm has an additional undelay step, where at each iteration an elementary inverse time-shift matrix must be applied, and as this algorithm also requires the most complex of the three search functions for locating the dominant coefficient, it is overall the most expensive per iteration. Due to the difficulty in comparing the computational complexity of the three algorithms, the computational time[3] to calculate the PQRD using each algorithm is recorded for each example and used to compare the three algorithms. The computational complexity of each algorithm is formulated in Appendix C.

[3]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

6.2.1 Example 1

For the first example, each of the three algorithms for calculating the PQRD was applied to the polynomial matrix

$$\underline{\mathbf{A}}_1(z) = \begin{bmatrix} 1+2z^{-1} & 2 & 2+z^{-1} \\ 3z^{-1} & 2+z^{-2} & 1+z^{-1} \\ 2 & 1+2z^{-2} & 2z^{-1} \end{bmatrix}. \tag{6.1}$$

For each algorithm the energy based truncation method, with $\mu = 10^{-6}$, was applied to both the paraunitary and the transformed polynomial matrices at the end of each iteration. For each implementation, the stopping condition was set so that each algorithm will stop once the magnitude of every coefficient associated with a polynomial element beneath the diagonal of the matrix is less than 10^{-3}. The results from applying each of the three algorithms to the polynomial test matrix $\underline{\mathbf{A}}_1(z)$ can be seen in Table 6.1.

This matrix initially has only four non-zero coefficients associated with polynomial elements positioned beneath the diagonal to drive to zero. However, each of the three algorithms will require more iterations and therefore also EPGRs than four to converge, as each element has an associated set of coefficients all of which will be affected by each application of an EPGR. For example, when applying the PQRD-BS algorithm to this polynomial matrix, the first two steps of the algorithm to drive all coefficients of the polynomial elements $\underline{a}_{21}(z) = 3z^{-1}$ and $\underline{a}_{31}(z) = 2$, to be sufficiently small, requires ten EPGRs. This is due to the non-zero lag coefficient of the polynomial element $\underline{a}_{11}(z) = 1 + 2z^{-1}$, which will also change under application of the EPGR forcing new coefficients to emerge in polynomial elements beneath the diagonal, but in the same column of the matrix. Note that the magnitude squared of the zero-lag coefficient of this polynomial element, i.e. the quantity $|a_{11}(0)|^2$, will increase monotonically throughout all iterations of the algorithm and so convergence is guaranteed.

Of the three algorithms, the PQRD-BS algorithm required the least number of EPGRs to converge, taking only 71 iterations and only one sweep of the algorithm to reach a point where all coefficients associated with polynomial elements beneath the diagonal of the transformed upper triangular polynomial matrix are less than 10^{-3} in magnitude. The Frobenius

	PQRD-BS	PQRD-BC	PQRD-SBR
Number of EPGRs	71	74	125
Number of Sweeps	1	1	-
Final order of $\underline{\mathbf{R}}(z)$	30	35	28
Final order of $\underline{\mathbf{Q}}(z)$	37	39	33
E_{rel}	3.53×10^{-3}	2.85×10^{-3}	2.62×10^{-3}
Final value of g	8.93×10^{-4}	8.66×10^{-4}	9.66×10^{-4}
Final value of L	2.46×10^{-3}	2.01×10^{-3}	3.54×10^{-3}
Computational Time (Seconds)	0.055	0.061	0.109

Table 6.1: Results from applying each of the three PQRD algorithms to the polynomial test matrix $\underline{\mathbf{A}}_1(z) \in \mathbb{R}^{3\times3\times2}$.

norm of the polynomial elements beneath the diagonal of the approximately upper triangular polynomial matrix obtained from this algorithm was found to be 2.46×10^{-3}, which is clearly very small when compared to the initial Frobenius norm of the polynomial matrix $\|\underline{\mathbf{A}}_1(z)\|_F = 6.56$. Figures 6.1 and 6.2 illustrate the approximately upper triangular polynomial matrix (of order 30) and the paraunitary transformation matrix (of order 37) obtained using this algorithm. From inspection of these figures, it can be observed that the series of coefficients for each polynomial element of the two matrices is approximately centred about the zero-lag coefficient. Furthermore, although the algorithms for calculating the PQRD only formulate an approximate decomposition, Figure 6.1 shows that the coefficients associated with polynomial elements beneath the diagonal are very small and so a good approximation has been calculated. The relative error of the decomposition was calculated according to equation (5.40) and was found to be 3.53×10^{-3}, which accounts for both the error observed from truncating the orders of both polynomial matrices and the fact that the algorithm only generates an approximately upper triangular polynomial matrix. Note that the order of both polynomial matrices generated by the algorithm could be further reduced if required for the application of the PQRD to MIMO communications by setting a higher value for the truncation parameter μ, but this will affect the accuracy of the decomposition performed and therefore also the relative error.

The performance of the PQRD-BC algorithm was similar to that of the PQRD-BS algo-

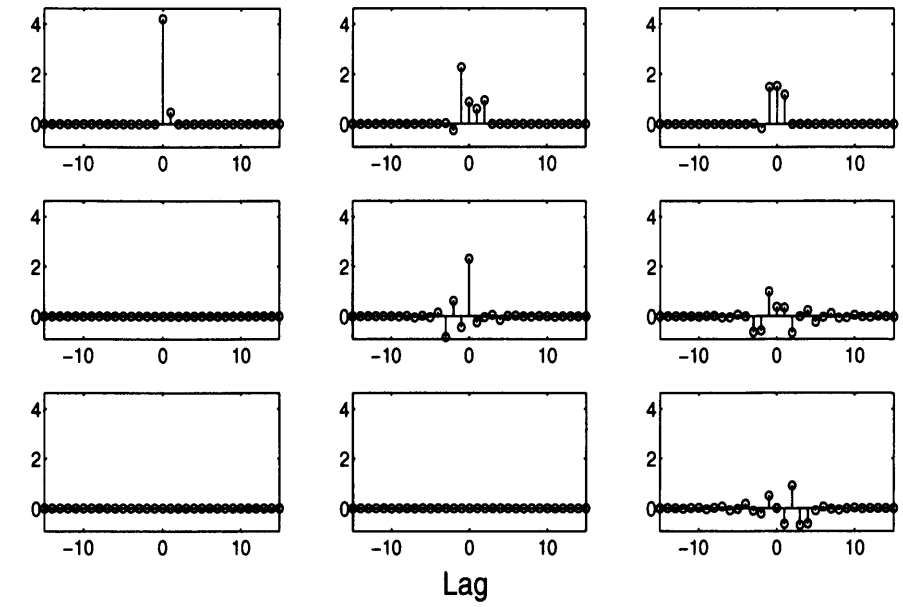

Figure 6.1: The coefficients of the polynomial elements of the approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BS algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}_1(z)$.

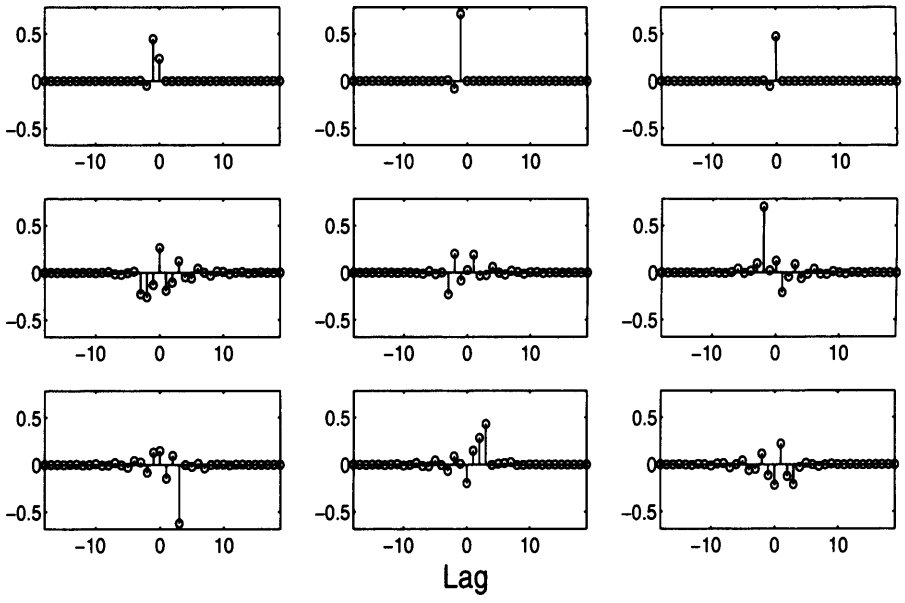

Figure 6.2: The coefficients of the polynomial elements of the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BS algorithm with polynomial input matrix $\underline{\mathbf{A}}_1(z)$.

rithm, although it did require a further three EPGRs to converge, therefore requiring more computational time. From Table 6.1, the orders of the two matrices obtained from this algorithm are seen to be slightly larger than those obtained using the PQRD-BS algorithm, although the relative error of this decomposition is smaller. The PQRD-SBR algorithm required the most EPGRs of the three algorithms, taking a total of 126 iterations to reach approximately the same level of convergence. Furthermore, in general each iteration of this algorithm is computationally more expensive than the other two algorithms, due to the additional inverse-time shift step and the more complicated search routine to locate the dominant coefficient. The computational time[4] for each of the three algorithms is contained in Table 6.1 and broadly reflects the varying number of EPGRs required for the different algorithms to converge.

Finally, Figure 6.3 demonstrates the Frobenius norm of all polynomial elements beneath the diagonal of the transformed polynomial matrix, at the end of each iteration, of each of the three algorithms. This measure, referred to as L, was previously defined in equation (5.41) of Chapter 5 and is a good measure to demonstrate the convergence of each of the three algorithms. In this figure, both the PQRD-BS and PQRD-BC algorithms can clearly be seen to converge in a process of steps, each requiring their own stopping criterion to be met before a subsequent step can begin. For this reason, the PQRD-SBR algorithm appears to initially converge faster, although if the stopping criterion ϵ is set smaller, this will no longer be the case. From this figure, it is also apparent that the measure L does not decrease monotonically, but this does not contradict the proofs of convergence detailed in Chapter 5.

[4]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

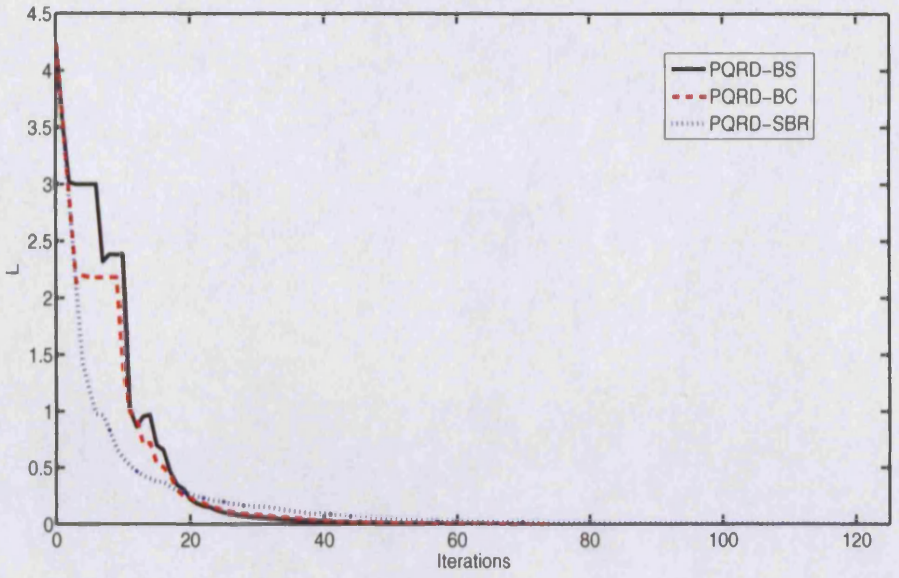

Figure 6.3: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix at each iteration of each of the algorithms for calculating the PQRD when applied to the first polynomial test matrix.

6.2.2 Example 2

For the second example, each of the three algorithms for calculating the PQRD was applied to the polynomial matrix $\underline{\mathbf{A}}_2(z) \in \mathbb{R}^{3\times3\times4}$, whose entries were fourth order FIR filters with real coefficients drawn independently from a normal distribution with mean zero and unit variance. This polynomial matrix has coefficient matrices $\mathbf{A}(0), \ldots, \mathbf{A}(4)$ and is illustrated in Figure 6.4. The polynomial matrix has 15 non-zero coefficients beneath the diagonal of the matrix, accounting for approximately 56% of the total Frobenius norm of the matrix.

Again each algorithm was applied to this polynomial matrix, using the same stopping condition and truncation parameter as used in the first example (i.e. $\epsilon = 10^{-3}$ and $\mu = 10^{-6}$). This time the PQRD-BC algorithm required the least amount of EPGRs (356) to converge to a point where all coefficients associated with the polynomial elements positioned beneath the diagonal of the approximately upper triangular polynomial matrix are less than 10^{-3} in magnitude. However, as with the previous example its performance was found to be similar to that of the PQRD-BS algorithm, which required only five additional EPGRs to reach

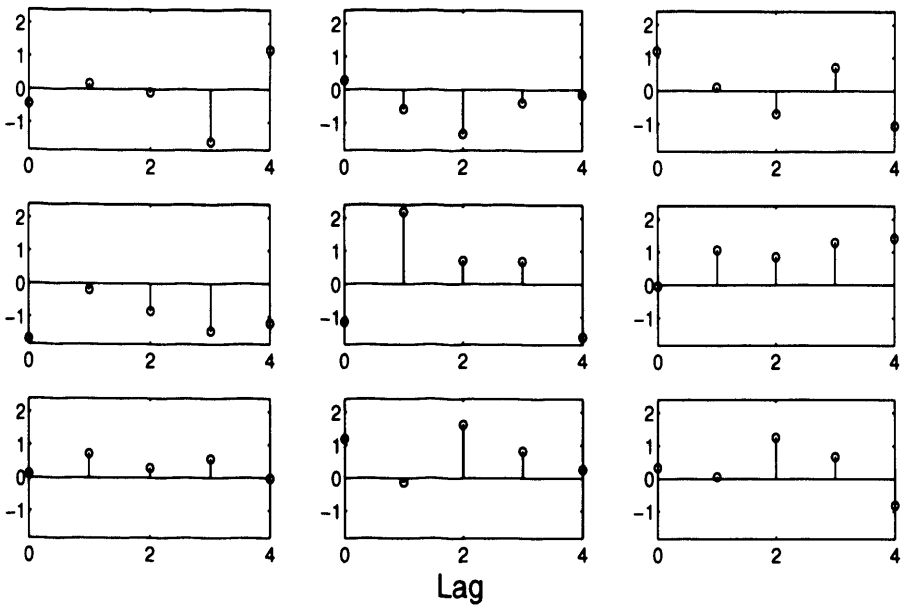

Figure 6.4: The polynomial matrix $\underline{\mathbf{A}}_2(z) \in \mathbb{R}^{3\times3\times4}$ to be used as input to each of the three algorithms for calculating the PQRD.

approximately the same level of decomposition. Once again, the PQRD-SBR approach can be seen to require considerably more EPGRs, requiring a total of 901 iterations to converge. The computational time for this algorithm is also approximately 10 times greater than that for the other two algorithms due to these additional iterations. The results from applying each of the three algorithms to the polynomial matrix $\underline{\mathbf{A}}_2(z)$ can be seen in Table 6.2, demonstrating that the PQRD-BC algorithm is clearly the best to use in this case. The PQRD-BC algorithm required the least time to converge and the orders of the resulting polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$ are both less than those obtained from the other two algorithms. The relative error for this decomposition was slightly larger than that observed from the other algorithms, but not by a significant amount. This measure has demonstrated that a very good approximate decomposition has been achieved with each of the PQRD algorithms.

The Frobenius norm of all polynomial elements beneath the diagonal of the polynomial transformed matrix at each iteration of each of the three decomposition algorithms can be seen in Figure 6.5. This measure, previously defined as L in equation (5.41) of Chapter 5, confirms the convergence of each of the three PQRD algorithms. Notice that this quantity does not

	PQRD-BS	PQRD-BC	PQRD-SBR
Number of EPGRs	361	356	901
Number of Sweeps	2	3	-
Final order of $\underline{\mathbf{R}}(z)$	111	66	128
Final order of $\underline{\mathbf{Q}}(z)$	113	77	135
E_{rel}	8.06×10^{-3}	9.13×10^{-3}	8.77×10^{-3}
Final value of g	9.87×10^{-4}	8.85×10^{-4}	9.92×10^{-4}
Final value of L	5.09×10^{-3}	4.08×10^{-3}	9.08×10^{-3}
Computational Time (Seconds)	0.425	0.411	4.565

Table 6.2: Results from applying each of the PQRD algorithms to the polynomial test matrix $\underline{\mathbf{A}}_2(z) \in \mathbb{R}^{3\times3\times4}$.

decrease monotonically for any of the algorithms and can be seen to abruptly increase by a considerable amount in the second sweep of both the PQRD-BS and PQRD-BC algorithms. Despite this, convergence of both algorithms is guaranteed and so at a future iteration of the algorithm, the quantity will be reduced again. This behaviour is due to the application of EPGRs allowing the Frobenius norm of the polynomial elements above the diagonal to be redistributed below the diagonal in columns positioned to the right of the coefficient that it is driving to zero. In fact, to zero the coefficient $a_{jk}(\tau)$ (where $j > k$ and $\tau \in \mathbb{Z}$ denotes the lag index), any coefficients associated with the polynomial elements $\underline{a}_{j,k+1}(z), \ldots, \underline{a}_{j,j-1}(z)$ can increase in magnitude squared, with the increase due to a decrease in the relevant elements above the diagonal. The quantity L can therefore increase at any stage (i.e. at any step of any sweep) of the algorithm, however, the most notable changes, such as the behaviour demonstrated in Figure 6.5, will only ever occur when multiple sweeps of either the PQRD-BS or PQRD-BC algorithms are required. The initial Frobenius norm of the matrix was found to be $\|\underline{\mathbf{A}}_2(z)\|_F = 6.39$, with the Frobenius norm of the elements beneath the diagonal equal to 3.58. Note that the measure L increases nearly to this value in the second sweeps of both the algorithms; the PQRD-BC algorithm to 2.83 and PQRD-BS algorithm suddenly increases to 3.33. However, rather than being distributed over all coefficients beneath the diagonal as in the input matrix $\underline{\mathbf{A}}_2(z)$, now the majority of this measure is contained in only one element and typically in only one of the associated coefficients, which can then be driven to zero by

application of a single EPGR.

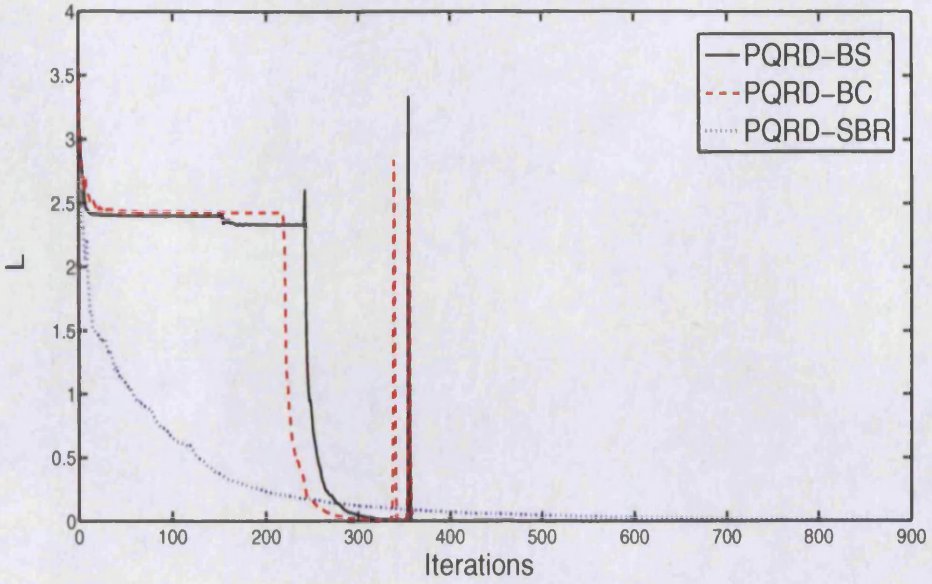

Figure 6.5: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix at each iteration of each of the algorithms for calculating the PQRD when applied to the polynomial matrix $\underline{\mathbf{A}}_2(z)$.

Figures 6.6 and 6.7 illustrate the approximately upper triangular and the paraunitary transformation polynomial matrices obtained using the fastest of the three algorithms to converge, the PQRD-BC algorithm. Notice that, unlike all other polynomial elements of the approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, the set of non-zero coefficients associated with the third diagonal polynomial element $\underline{r}_{33}(z)$ is not centred about the zero-lag coefficient. The same observation can also be made about the coefficients associated with the polynomial elements of the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$. Although this is not necessarily a problem, if all of the polynomial elements are forced to be centred about the same lag by applying an elementary delay matrix, then the order of both polynomial matrices can generally be reduced further by applying the energy based truncation method once more. Furthermore, this final step will generally not reduce the accuracy of the decomposition any further and the overall transformation will remain paraunitary.

The polynomial elements of the matrix become out of alignment due to multiple applica-

tions of elementary delay matrices, which are applied over a series of EPGRs. For example, to drive any of the coefficients associated with the polynomial element $\underline{a}_{31}(z)$ sufficiently small, all polynomial elements of the matrix in the same row as this element will be affected by the elementary delay matrix incorporated within the required EPGR. Over many iterations of the algorithm these operations will lead to polynomial elements in the third row, which are not all centred over the same series of lags as all other elements of the polynomial matrix. However, the non-zero coefficients associated with all elements in each row of the matrix, will generally be positioned over the same series of lags. However, if there are polynomial elements positioned beneath the elements that are not aligned in the matrix, i.e. if there was a fourth row to the polynomial matrix used in this example, then over future iterations to zero polynomial elements positioned beneath these non-aligned elements, the appropriate EPGRs will result in the polynomial elements in the third row being realigned to be centred about the zero-lag. Therefore, the only possible polynomial elements, where this realignment cannot happen, will be positioned in the bottom row containing a diagonal polynomial element of the matrix and this is therefore generally only ever an issue with input matrices that are either square or fat, i.e. for matrices that have at least as many columns as rows. The same problem can also be seen in the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$, whose elements will also not be aligned, in accordance to the elements of the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$.

The polynomial elements of the resulting approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ obtained from the PQRD-BC algorithm, depicted in Figure 6.6, can easily be realigned by applying a series of elementary delay matrices to this matrix, such that the coefficient with the largest magnitude in each diagonal polynomial element becomes the coefficient of z^0. Note that if using a final alignment, the delay matrices must also be applied to the paraunitary polynomial matrix $\underline{\mathbf{Q}}(z)$. For this example, aligning the associated series of coefficients, of the polynomial elements of both matrices obtained using the PQRD-BC algorithm, to be over the same series of lags and then implementing a final truncation of these matrices using the same value of the truncation parameter, i.e. $\mu = 10^{-6}$, the order of $\underline{\mathbf{R}}(z)$ was reduced from 66 to 51 and the order of $\underline{\mathbf{Q}}(z)$ from 77 to 60, with no additional cost

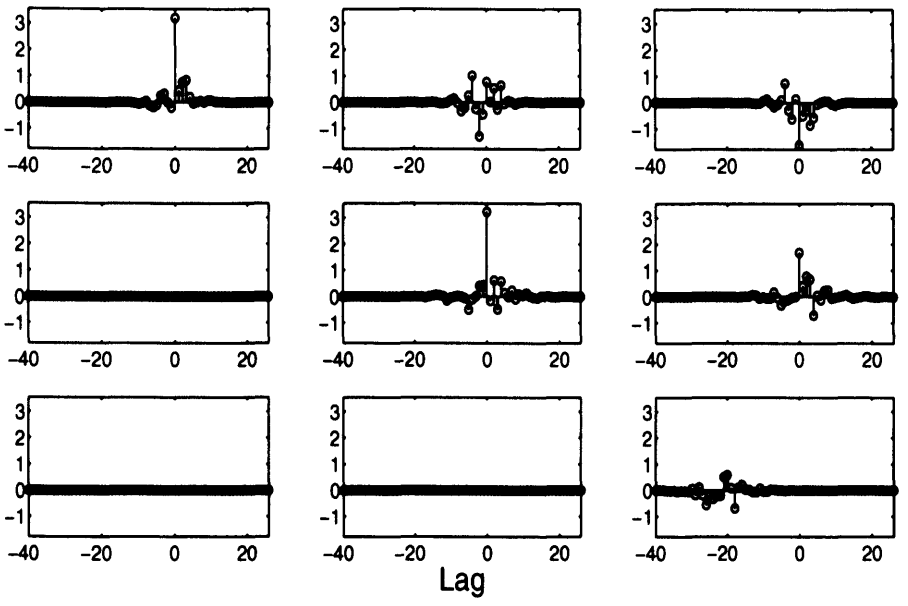

Figure 6.6: The coefficients of the polynomial elements of the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BC algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}_2(z)$.

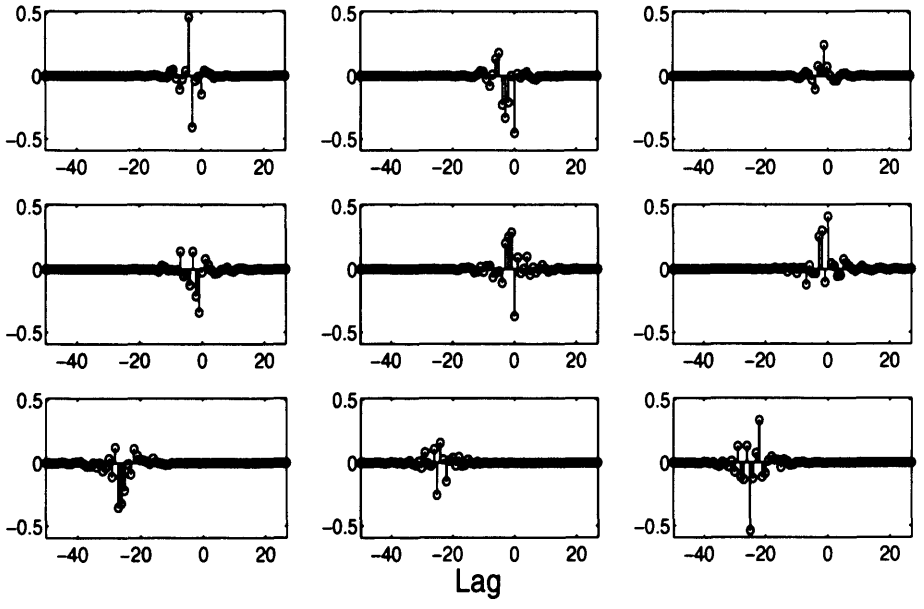

Figure 6.7: The coefficients of the polynomial elements of the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BC algorithm with polynomial input matrix $\underline{\mathbf{A}}_2(z)$.

to the relative error.

Moreover, if this additional step is undertaken at regular intervals in each algorithm it will also ensure that it is not possible to truncate a significant number of coefficients associated with just one of the polynomial elements from the matrix, such as $\underline{r}_{33}(z)$ in this example. It could even be possible to truncate an entire polynomial element from the matrix if this additional step is not undertaken, although it would probably also require a inappropriate value of μ. Note that this was not an issue within the SBR2 algorithm, as the elementary delay matrices are applied on both the left and right hand-side of the polynomial para-Hermitian input matrix, which ensures that the diagonal elements are not affected by the application of elementary delay matrices and so the polynomial transformed matrix remains centred about the zero-lag coefficient matrix at all times. Furthermore, note that this point is rarely an issue with the PQRD-SBR algorithm, due to the elementary inverse time-shift step following the application of the EPGR, which will help keep all polynomial elements centred about the zero-lag coefficient matrix. If this step is undertaken throughout the algorithm it will be referred to as the Zero-Lag Specification Step (ZLSS), which will now be defined.

6.2.3 Zero-Lag Specification Step

The significance of the zero-lag coefficient matrix has been discussed in Section 6.1. This coefficient matrix plays a pivotal role in the proof of convergence for each of the three algorithms, as each algorithm converges through the columns of the polynomial matrix from left to right, even in the SBR approach where there is no specified order in which the EPGRs are applied. Each column converges as a result of the coefficient associated with the zero-lag of the diagonal element in the column increasing in magnitude squared to account for the decrease in the Frobenius norm of the elements beneath the diagonal, i.e. the k^{th} column converges as a result of the quantity $|a_{kk}(0)|^2$ increasing. During this process, the application of EPGRs will affect all other coefficients associated with the diagonal element in the column and these coefficients can decrease in magnitude squared, which will then also mean the appropriate coefficients beneath the diagonal increasing in magnitude squared. It is therefore reasonable to insist on the diagonal zero-lag coefficient being the largest possible coefficient,

which is easily enforced by applying an elementary delay matrix of the form demonstrated in equation (3.8), where k defines the row and column index of the diagonal element and t the lag index of the coefficient with maximum magnitude in this element.

This step will be referred to as the Zero-Lag Specification (ZLS) step and aims to realign the series of coefficients associated with each diagonal element of the transformed polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$, to ensure that the zero-lag coefficient is the largest coefficient in magnitude squared. This is achieved by applying a series of elementary delay matrices, formulated according to Section 3.3.2 and can be implemented at the beginning of each iteration (including the first) within any of the three algorithms for calculating the PQRD. Suppose the set of coefficients with largest magnitude in the diagonal polynomial elements of $\underline{\mathbf{A}}(z)$ is found to be $\{a_{11}(t_1), a_{22}(t_2), \ldots, a_{NN}(t_N)\}$, where $N = \min(p, q)$. Then the appropriate delay matrices are applied to $\underline{\mathbf{A}}(z)$ to obtain the transformed polynomial matrix

$$\underline{\mathbf{A}}'(z) = \underline{\mathbf{B}}^{(N,t_N)}(z) \ldots \underline{\mathbf{B}}^{(1,t_1)}(z)\underline{\mathbf{A}}(z). \tag{6.2}$$

Following this transformation, all coefficients of the polynomial matrix will be centred about the zero-lag coefficient matrix. Note that this additional step will increase the computational load of the algorithm and so it is important to see if the additional computational cost is more or less than the computational cost observed if the number of iterations of any of the algorithms is reduced. As this is not easily assessed, due to the dependence of the computational cost upon the order of the two matrices within the algorithm at each iteration, the computational times for the two implementations of each algorithm are compared. Note that if the ZLS step is implemented within the PQRD-SBR algorithm, then the third step (referred to as the un-delay or inverse time-shift step in the description of the algorithm in Section 5.6) is no longer necessary. In this case, the ZLS step is sufficient to remove the behaviour demonstrated by Figure 5.10 and is therefore unnecessarily adding to the computational time if it is used.

The two possible advantages of applying this ZLS step are: the orders of the polynomial matrices can be truncated with barely any deterioration in the relative error of the decompo-

sition, which is extremely useful for the application of the PQRD to MIMO communications where the order of the matrices is of critical importance; secondly, if the ZLS step is performed between every iteration in each of the algorithms it could ensure that a more accurate polynomial matrix decomposition is obtained, although this is not guaranteed. The drawback with implementing this technique at each iteration is that it will add to the computational cost of the associated algorithm. This additional realignment of the polynomial elements at the beginning of each iteration of the algorithm is now assessed in the context of the previous example.

6.2.4 Example 2 Continued

The three algorithms for calculating the PQRD were again applied to $\underline{\mathbf{A}}_2(z)$, this time implementing the ZLS step at the start of each iteration of each of the algorithms and the results obtained are contained in Table 6.3. From these results, the relative error observed has improved for each of the three decompositions, whist the order of the resulting polynomial matrices $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{Q}}(z)$ are, in each case, considerably shorter. Furthermore, the number of EPGRs required, and therefore also the computational time, in both the PQRD-BS and the PQRD-SBR algorithm have been reduced despite the additional computations required to implement the ZLS step. In fact, the PQRD-SBR algorithms required an impressive 528 fewer iterations, reducing the computational time from 4.57 to 1.19 seconds. The PQRD-BS algorithm also required fewer iterations, although the computational time increased. Similarly, the PQRD-BC algorithm did not improve by implementing the ZLS step, requiring a further 40 EPGRs. In this case, the performance of the decomposition algorithm has improved in terms of the relative error and the order of the matrices. Note however, that the original PQRD-BC algorithm as detailed in Chapter 5 was the fastest of the three algorithms to converge.

Finally, Figure 6.8 illustrates the Frobenius norm of all polynomial elements beneath the diagonal of the transformed polynomial matrix at the end of each iteration, for each of the three decomposition algorithms. This figure displays the same measure for each of the algorithms, with and without the ZLS step, and shows that the erratic behaviour observed in

	PQRD-BS	PQRD-BC	PQRD-SBR
Number of EPGRs	330	402	373
Number of Sweeps	1	2	-
Final order of $\underline{\mathbf{R}}(z)$	50	46	69
Final order of $\underline{\mathbf{Q}}(z)$	59	43	43
E_{rel}	6.23×10^{-3}	8.05×10^{-3}	7.72×10^{-3}
Computational Time (Seconds)	0.52	1.61	1.19

Table 6.3: Results from applying each of the algorithms for calculating the PQRD to the polynomial test matrix $\underline{\mathbf{A}}_2(z) \in \mathbb{R}^{3\times3\times4}$ whilst using the zero-lag specification step.

the second sweeps of the original PQRD-BS and the PQRD-BC algorithms can be removed by implementing the ZLS step at the start of each iteration.

Alternatively, the jumpy behaviour of the measure L observed in Figure 6.5 when applying the PQRD-BS and the PQRD-BC algorithms could have been removed by applying an inverse time-shift matrix as demonstrated by equation (5.30) to the transformed polynomial matrix after every application of an EPGR. This was also implemented at the end of each iteration of the PQRD-SBR algorithm; however, in the two algorithms that proceed as a series of steps this will only need to be applied in any iterations undertaken following the first sweep of either of these algorithms. Note that for this example, implementing the undelay step, the PQRD-BS algorithm required 376 EPGRs and took 1.71 seconds, whilst the PQRD-BC algorithm required 398 iterations and 1.57 seconds to converge. For this example, the best algorithm to use is the PQRD-BS with the ZLS step. The computational time for this algorithm was not as small as that of the PQRD-BC algorithm without the ZLS step, but the erratic behaviour in the convergence was not observed.

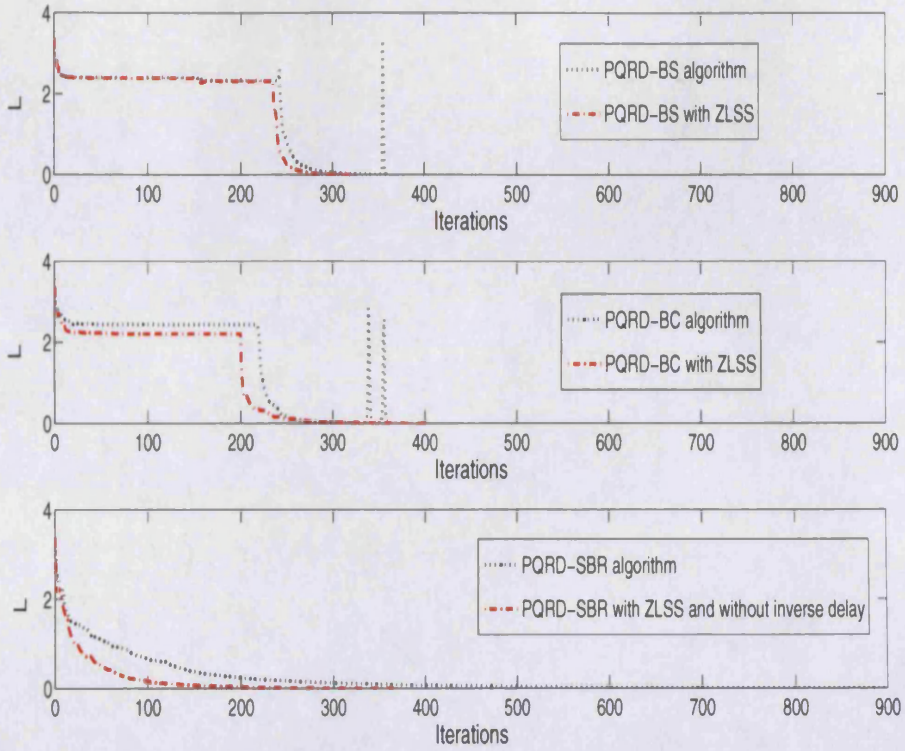

Figure 6.8: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix over the series of iterations for each of the PQRD algorithms when applied to $\underline{\mathbf{A}}_2(z)$ for the cases (i) using the original code as described in Chapter 5 and (ii) when implementing the algorithms with the ZLSS.

6.2.5 Example 3

For the third example, the polynomial matrix $\underline{\mathbf{A}}_3(z) \in \mathbb{C}^{5\times3\times4}$ illustrated in Figure 6.9, was chosen to be complex, where both the real and imaginary parts of each polynomial element were independently drawn from a normal distribution with mean zero and unit variance. This polynomial matrix has complex coefficient matrices $\mathbf{A}(0),\ldots,\mathbf{A}(4)$. The Frobenius norm of this matrix was found to be 11.42, with 79% of this positioned beneath the diagonal of the matrix.

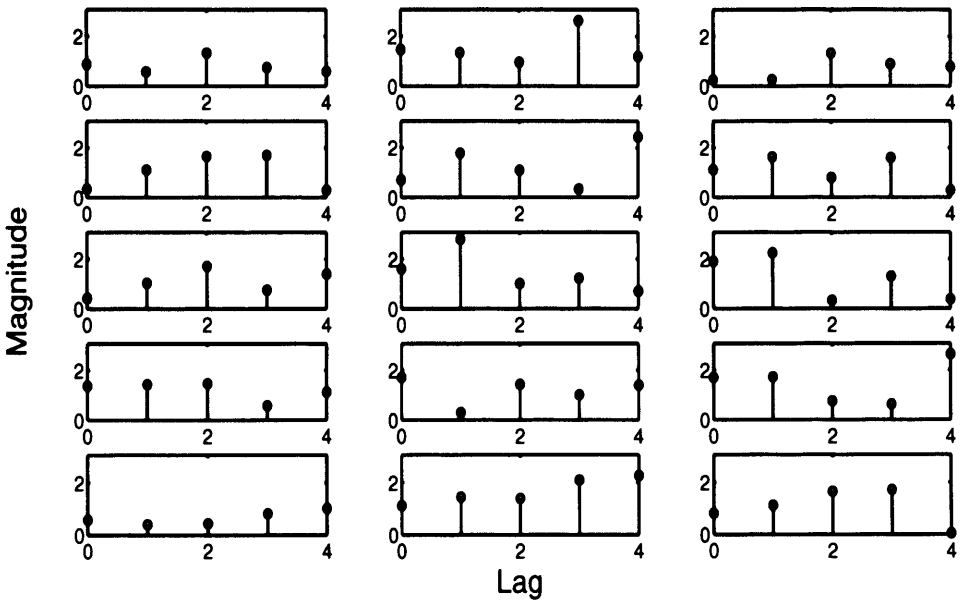

Figure 6.9: The magnitude of the coefficients of the polynomial elements of the polynomial matrix $\underline{\mathbf{A}}_3(z)$ to be used as input to each of the three algorithms for calculating the PQRD.

Each of the three original algorithms for calculating the PQRD (as described in Chapter 5) were applied to the polynomial matrix $\underline{\mathbf{A}}_3(z)$. The energy based truncation method was again used within each algorithm with $\mu = 10^{-6}$ and the stopping criterion set as $\epsilon = 10^{-3}$. The results from applying each of the three algorithms to this polynomial matrix are contained in Table 6.4. The results confirm that the PQRD-BC algorithm was again significantly faster than the two other algorithms to implement, requiring only 589 EPGRs to converge to an approximately upper triangular polynomial matrix, where the magnitude

of each coefficient associated with any of the polynomial elements positioned beneath the diagonal is less than 10^{-3}. Furthermore, this algorithm required the least time to converge and the orders of the resulting matrices from this algorithm are smaller than those obtained from the other two decomposition algorithms. The relative error of the decomposition obtained using this algorithm was found to be 9.53×10^{-3}, illustrating that a very good approximate decomposition can be achieved. Figures 6.10 and 6.11 illustrate the polynomial matrices $\underline{\mathbf{Q}}(z)$ (of order 43) and $\underline{\mathbf{R}}(z)$ (of order 43) obtained using the PQRD-BC algorithm. From inspection of the series of coefficients of the polynomial elements of the polynomial matrix $\underline{\mathbf{R}}(z)$ in Figure 6.10, it clear that a good approximation to an upper triangular polynomial matrix has been made by the decomposition algorithm.

The PQRD-BS algorithm required considerably more EPGRs (762) than the PQRD-BC algorithm and as a result slightly more computational time. As with the previous two examples, the PQRD-SBR algorithm was the slowest to converge, requiring over twice as many EPGRs than either the PQRD-BS or PQRD-BC algorithms. As a result, the computational time was much slower, taking over four times longer than either of the other two algorithms to converge.

	PQRD-BS	**PQRD-BC**	**PQRD-SBR**
Number of EPGRs	762	583	1607
Number of Sweeps	2	2	-
Final order of $\underline{\mathbf{R}}(z)$	34	33	43
Final order of $\underline{\mathbf{Q}}(z)$	50	43	48
E_{rel}	7.65×10^{-3}	9.53×10^{-3}	5.94×10^{-3}
Final value of g	9.28×10^{-4}	9.42×10^{-4}	9.96×10^{-4}
Final value of L	6.26×10^{-3}	5.58×10^{-3}	1.03×10^{-2}
Computational Time (Seconds)	0.87	0.69	4.07

Table 6.4: Results from applying each of the PQRD algorithms to the polynomial test matrix $\underline{\mathbf{A}}_3(z) \in \mathbb{C}^{5\times3\times4}$.

The results were again repeated, applying each algorithm to the polynomial matrix $\underline{\mathbf{A}}_3(z)$. However, this time the ZLS step was applied at the start of each iteration of each of the three algorithms. The results obtained when using the ZLS step can be seen in Table 6.5. As

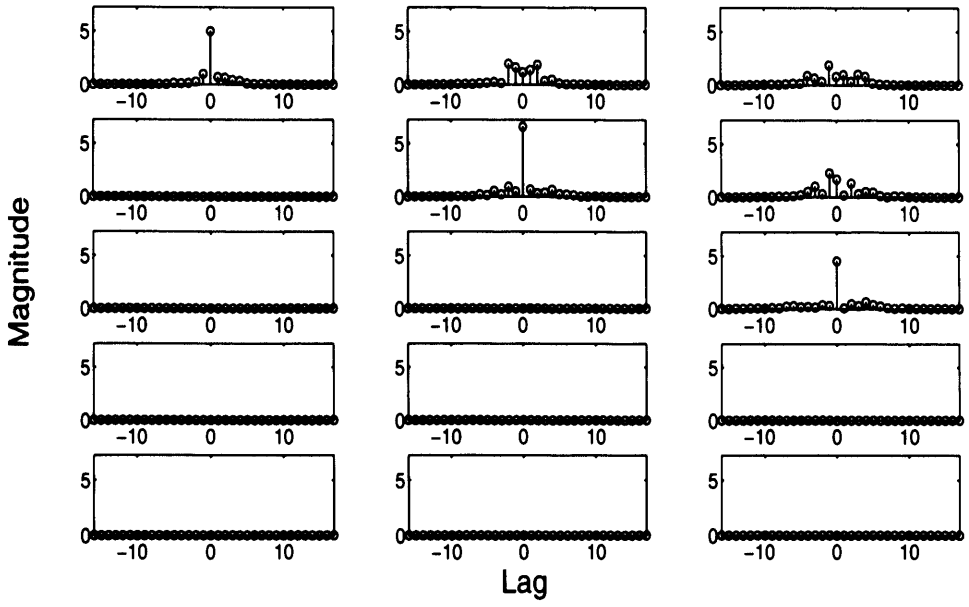

Figure 6.10: The coefficients of the polynomial elements of the approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BC algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}_3(z)$.

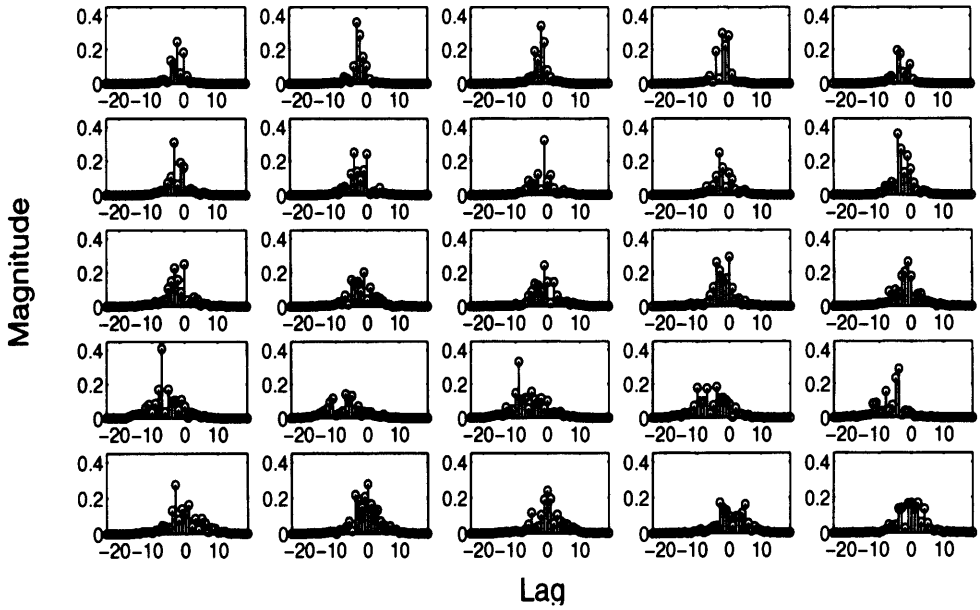

Figure 6.11: The coefficients of the polynomial elements of the paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BC algorithm with polynomial input matrix $\underline{\mathbf{A}}_3(z)$.

with the previous example, the most notable improvement can be seen in the PQRD-SBR algorithm, which required 454 fewer iterations to converge. However, the computational time required to calculate this decomposition has increased, demonstrating that implementing the additional step was computationally more expensive that the extra iterations required if the ZLS step is not used. The number of EPGRs have also decreased for the PQRD-BS algorithm, however, the computational time has again increased. Finally, for the PQRD-BC algorithm the number of EPGRs required for the algorithm to converge has increased. Figure 6.12 illustrates the Frobenius norm of all polynomial elements beneath the diagonal of the transformed polynomial matrix at the end of each iteration, for each of the three decomposition algorithms. This figure displays the same measure for each of the algorithms, with and without the ZLS step. From this figure, it can be seen that there was no erratic behaviour observed in the second sweep of either the PQRD-BS or the PQRD-BC algorithms and so in this respect the ZLS step offers no advantage over the original algorithms.

Although the ZLS step demonstrated some potential advantages when it was applied as part of the three algorithms for calculating the PQRD of $\underline{\mathbf{A}}_2(z)$, for this example the additional step offers no advantage to any of the PQRD algorithms. The number of iterations required for the PQRD-BS and the PQRD-BC algorithms to converge was reduced, but the additional step forced the computational time for each of these algorithms to increase. Furthermore, for this example the ZLS step did not reduce the orders of the two polynomial matrices generated by any of the algorithms or help reduce the relative error. For this example, the PQRD-BC algorithm as detailed in Chapter 5 requires the least number of EPGRs and computational time to converge. This algorithm is therefore the most efficient for this example.

	PQRD-BS	PQRD-BC	PQRD-SBR
Number of EPGRs	650	602	1153
Number of Sweeps	2	2	-
Final order of $\underline{\mathbf{R}}(z)$	30	47	45
Final order of $\underline{\mathbf{Q}}(z)$	45	43	51
E_{rel}	7.71×10^{-3}	8.96×10^{-3}	7.69×10^{-3}
Computational Time (Seconds)	0.95	1.36	4.36

Table 6.5: Results from applying each of the PQRD algorithms to the polynomial test matrix $\underline{\mathbf{A}}_3(z) \in \mathbb{C}^{5\times3\times4}$, each implementing the ZLS step at each iteration of the algorithm.

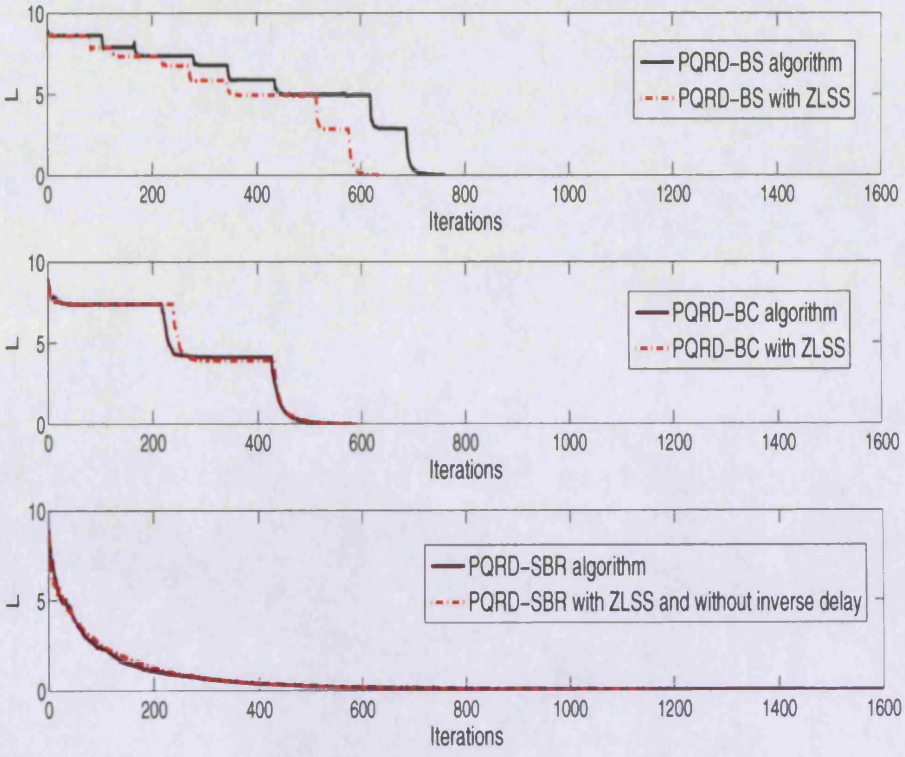

Figure 6.12: The Frobenius norm of the polynomial elements beneath the diagonal of the transformed polynomial matrix over the series of iterations for each of the PQRD algorithms when applied to $\underline{\mathbf{A}}_3(z)$ for the cases (i) using the original code as described in Chapter 5 and (ii) when implementing the algorithms with the ZLSS.

6.2.6 Example 4

For the final example, the three algorithms for calculating the PQRD were applied to the polynomial matrix $\underline{\mathbf{A}}_4(z) \in \mathbb{C}^{5\times3\times4}$. This matrix is the same as the polynomial matrix $\underline{\mathbf{A}}_3(z)$, except that the third column of the matrix has been set equal to the second, resulting in a polynomial matrix that does not have full column rank. As the PQRD-BC algorithm converged in the least number of iterations when applied to $\underline{\mathbf{A}}_3(z)$, this algorithm was again used to calculate the PQRD of $\underline{\mathbf{A}}_4(z)$. This algorithm required 387 iterations and only one sweep of the algorithm to converge to a point where $g < 10^{-3}$. The upper triangular matrix obtained by the algorithm is illustrated in Figure 6.13, where the third diagonal polynomial element of the matrix $\underline{r}_{33}(z)$ is approximately equal to zero. In fact,the Frobenius norm of this polynomial element is found to be 2.66×10^{-6}. The paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained from the decomposition is demonstrated in Figure 6.14.

Note that the PQRD of any polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ exists even if the matrix is not of full rank. However, in this case, a number of the diagonal elements of the polynomial matrix $\underline{\mathbf{R}}(z)$ will be equal to zero. For the scalar matrix case, this will mean that if the decomposition is to be used to solve a set of linear equations of the form $\mathbf{Ax} = \mathbf{b}$ (for $\mathbf{x}$, given $\mathbf{A}$ and $\mathbf{b}$) by back substitution, then it will not be possible to calculate every element of $\mathbf{x}$. Depending on the position of the zero(s) it may still be possible to obtain estimates of some of the elements. The same will be true if this problem is extended to polynomial matrices.

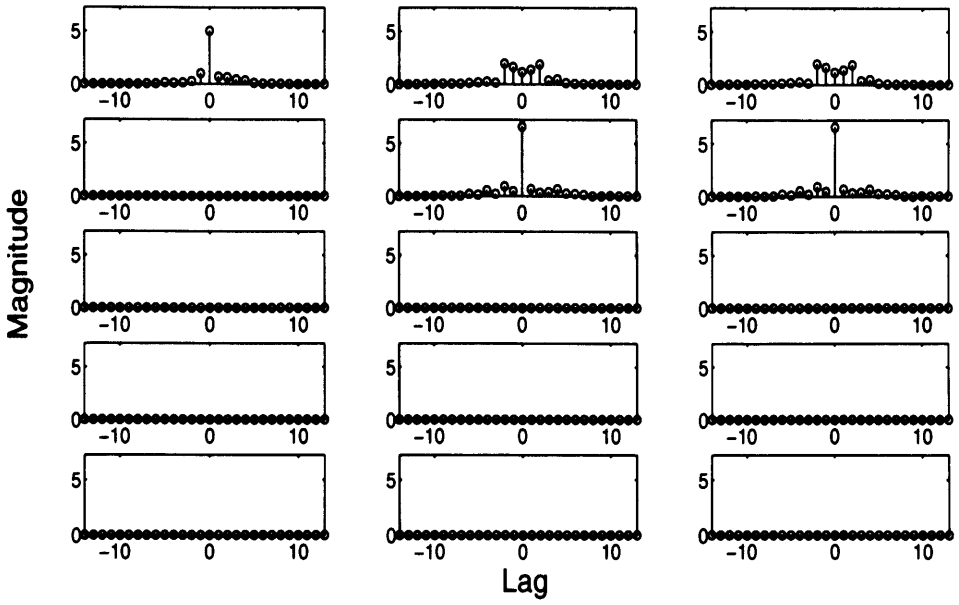

Figure 6.13: The coefficients of the polynomial elements upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, obtained when the PQRD-BC algorithm was applied to the polynomial matrix $\underline{\mathbf{A}}_4(z)$.

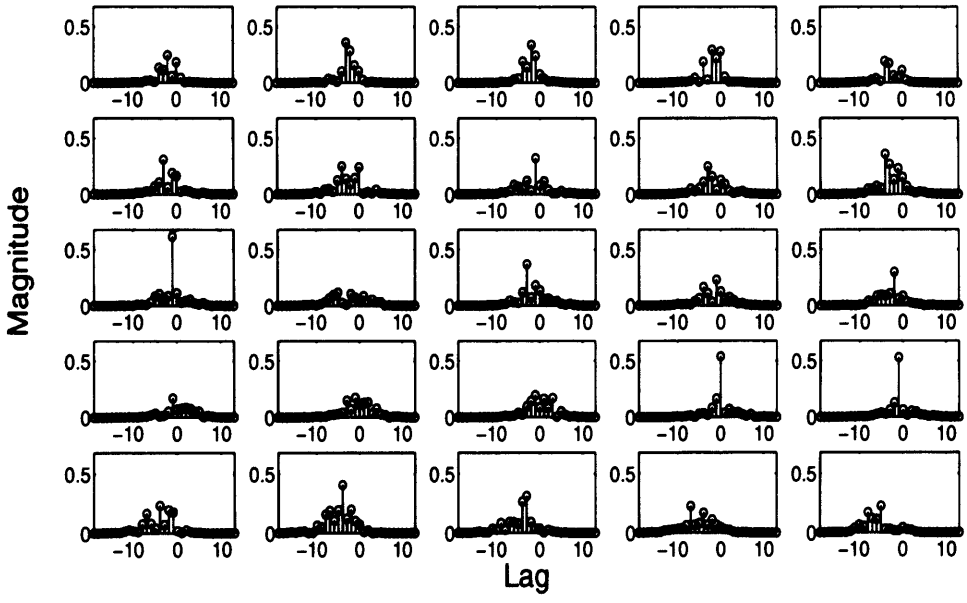

Figure 6.14: The coefficients of the polynomial elements paraunitary transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BC algorithm with polynomial input matrix $\underline{\mathbf{A}}_4(z)$.

6.2.7 Other Zero-Lag Options

In Section 6.1 the importance of the zero-lag coefficient matrix was discussed. Within each of the PQRD algorithms, the zero-lag coefficient matrix can be chosen from any of the coefficient matrices of the polynomial matrices. However, due to the non-uniqueness of the polynomial decomposition, the choice will affect the decomposition performed. In particular, the choice will result in the different algorithms requiring a different number of EPGRs to converge. Furthermore, it will also result in polynomial matrices of different orders. The choice of the zero-lag coefficient matrix is important for convergence of each of the algorithms. For this reason, three methods for choosing the zero-lag coefficient matrix from the polynomial input matrix at the beginning of each iteration of the algorithms has been developed. The three techniques are

1. **Method 1:** The zero-lag coefficient matrix is specified to be the first coefficient matrix within each example. (This has been used for all examples in this chapter)
2. **Method 2:** The zero-lag coefficient matrix is chosen to be the coefficient matrix associated with the coefficient of $\underline{a}_{11}(z)$ with largest magnitude.
3. **Method 3:** The zero-lag coefficient matrix is chosen to be the coefficient matrix with the largest Frobenius norm.

Each of the three PQRD algorithms (as detailed in Chapter 5) was again applied to the first three examples from this chapter, using each of the three options discussed above. The number of EPGRs required for each implementation of each of the three algorithms to converge are contained in Table 6.6. In this table, for each algorithm and each example, the option that required the least number of EPGRs is highlighted in red.

The results suggest that for both the PQRD-BS and PQRD-BC algorithms, there is no relationship between either the initial choice of the zero-lag coefficient matrix and the number of EPGRs required for the algorithm to converge. However, with the PQRD-SBR algorithm choosing the zero-lag coefficient matrix to be the coefficient matrix containing the largest coefficient of the polynomial element $\underline{a}_{11}(z)$, i.e. using the second method, will reduce the number of EPGRs required significantly for each of the three examples. For the second example, the PQRD-SBR algorithm using the second method requires fewer EPGRs than the

	PQRD-BS			PQRD-BC			PQRD-SBR		
	M.1	M.2	M.3	M.1	M.2	M.3	M.1	M.2	M.3
Example 1	**71**	85	85	74	**59**	**59**	125	**104**	**104**
Example 2	361	333	**324**	356	414	**350**	901	**294**	479
Example 3	**762**	792	788	**583**	627	585	1607	**1318**	1367

Table 6.6: The number of EPGRs required for each algorithm for calculating the PQRD to converge, when the initial zero-lag coefficient matrix is specified using the three different options method 1 (denoted as M.1 in the table), method 2 (M.2) or method 3 (M.3). For each example, the method requiring the least number of EPGRs for each algorithm is highlighted in red.

PQRD-BC algorithm, which currently offers the best performance. However, each iteration of the PQRD-SBR algorithm is typically computationally more expensive than a single iteration of the PQRD-BC algorithm and so the computational time required for the decomposition is still larger than that observed with the PQRD-BC algorithm.

6.3 Conclusions

This chapter has demonstrated the performance of the three algorithms for calculating the QRD of a polynomial matrix through several examples. The results have shown that the PQRD-BS and PQRD-BC algorithms generally require considerably fewer EPGRs than the PQRD-SBR algorithm. The SBR approach, although an extremely efficient technique for achieving the EVD of a polynomial matrix, is not the most appropriate for calculating the PQRD. In hindsight, this could be expected due to the polynomial input matrix converging to an upper triangular matrix through the columns of the matrix from left to right and so a process of steps, which will transform the matrix by annihilating the elements of the matrix from left to right, would be more appropriate for convergence. Furthermore, typically the PQRD-BC algorithm required the least number of EPGRs and for the single example where this is not the case (Example 1), this algorithm only required a few more iterations to converge than the PQRD-BS algorithm. Note that if both these algorithms are applied to a scalar matrix they will require the same number of Givens rotations, but the PQRD-

BC algorithm will be computationally slightly more expensive, due to the routine required to find the dominant coefficient in each step. However, when applying the algorithms to a polynomial matrix it becomes more difficult to compare the algorithms, due to the additional dimension to the problem resulting from each polynomial element of the matrix having an associated set of coefficients, which must all be driven to zero to achieve the decomposition. Furthermore, the computational cost of each algorithm is entirely dependent on the order of the polynomial matrices throughout the algorithm and the orders of both matrices, which will grow at each iteration, cannot be specified in advance.

The significance of the zero-lag coefficient matrix within the PQRD has been discussed within this chapter. This coefficient matrix is fundamental for convergence of each of the algorithms and whichever coefficient matrix is chosen to be the zero-lag coefficient matrix from the polynomial input matrix to the algorithm, the choice will affect the decomposition performed. In particular, the choice of this coefficient matrix will affect the numbers of iterations required for the algorithm to converge and will also produce a different paraunitary transformation and approximately upper triangular polynomial matrices of different orders. The output polynomial matrices from the algorithm are not unique, but this is not an issue for the potential application of the algorithm discussed in Chapter 8.

Several methods for choosing the zero-lag coefficient matrix from the polynomial input matrix have been discussed in this chapter. However, the basic PQRD-BC algorithm as detailed in Chapter 5, without any additional step, appears to be the best choice of algorithm to calculate a QRD of a polynomial matrix. The results have demonstrated that the choice of zero-lag coefficient matrix can significantly improve the performance of the PQRD-SBR algorithm and can drastically reduce the number of iterations required for this algorithm to converge. A process of realigning the zero-lag coefficient matrix of the input matrix has been presented in Section 6.2.3 and for some examples, using this technique, can reduce the number of iterations required for each algorithm. However, using this technique requires additional computations and so it does not always provide improved performance, even if the number of EPGRs has been vastly reduced. This step also removes any erratic behaviour observed in the measure of the Frobenius norm of the polynomial elements beneath the diagonal of

the matrix over the series of iterations (i.e. EPGRs) of the PQRD-BS and the PQRD-BC algorithms that is often observed if multiple sweeps of either of these algorithms are required. This behaviour can be seen in Figure 6.5. The choice of the zero-lag coefficient matrix adds another parameter to each of the algorithms, which is not present for the scalar matrix decomposition. Finally, note that this point was never an issue in the SBR2 algorithm as the input matrix was para-Hermitian and must remain so through all iterations of the algorithm.

Chapter 7

The Singular Value Decomposition of a Polynomial Matrix

7.1 Introduction

The singular value decomposition of a scalar matrix is one of the most useful developments in linear algebra. It can be used to determine the rank, range and null space of a matrix, but can also be used for calculating the pseudo-inverse of a matrix and for solving a set of homogeneous linear equations. Consequently, the decomposition has a wide range of applications in areas such as automatic control, scientific computing and narrowband adaptive sensor array processing [25]. In the context of signal processing, it can be used to decorrelate a set of instantaneously mixed signals and often as a result can be used to identify and then separate the signal and noise subspaces. The SVD is also the method of choice for solving most linear least-squares problems as it offers a numerically robust solution, which can be calculated by directly applying the SVD to the data matrix [25,38]. There exist several techniques for calculating the decomposition, which are discussed along with a detailed description of the decomposition and its properties in Chapter 2.

However, in broadband signal processing where polynomial matrices are generally part of the generative model for the observations, the SVD can no longer be applied as each element of the matrix now consists of a polynomial with an associated set of coefficients. Instead, a polynomial matrix singular value decomposition (PSVD) has been developed. For the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$, the objective of the PSVD is to calculate the paraunitary

matrices $\underline{\mathbf{U}}(z) \in \underline{\mathbb{C}}^{p \times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q \times q}$ such that

$$\underline{\mathbf{U}}(z)\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{V}}}(z) = \underline{\mathbf{S}}(z) \tag{7.1}$$

where $\underline{\mathbf{S}}(z) \in \underline{\mathbb{C}}^{p \times q}$ denotes a diagonal polynomial matrix and $\widetilde{(\cdot)}$ the paraconjugation of the polynomial matrix. The coefficients of the polynomial elements of $\underline{\mathbf{A}}(z)$ can be either real or complex, however, the advantage of this decomposition over the polynomial matrix eigenvalue decomposition (PEVD) is that it can be applied to polynomial matrices of any dimension, i.e. where it is not required that $p = q$ for the polynomial matrices in equation (7.1). As a consequence of using paraunitary matrices the transformation will be energy (or norm) preserving and so

$$\|\underline{\mathbf{A}}(z)\|_F^2 = \|\underline{\mathbf{S}}(z)\|_F^2 . \tag{7.2}$$

This chapter discusses two methods of obtaining a singular value decomposition of a polynomial matrix (PSVD). Firstly, the PSVD by polynomial matrix QR decomposition (PQRD) algorithm is introduced. This algorithm operates by iteratively applying two PQRDs to the polynomial matrix $\underline{\mathbf{A}}(z)$ to transform it into a diagonal polynomial matrix $\underline{\mathbf{S}}(z)$. Secondly, an already existing generalisation of the SVD to polynomial matrices is briefly discussed for comparison purposes. This method was introduced in [21] and operates by calculating two PEVDs, each formulated using the SBR2 algorithm, to calculate the parauntiary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$. A potential application of the decomposition lies in the area of MIMO communications, where it can be used to transform a MIMO system into a set of independent subchannels. Recent research, which is discussed further in the penultimate chapter of this thesis, has shown the PSVD by PEVD approach for this application obtains a good average bit error rate (BER) performance for transmission over frequency selective quasi-static channels [11–14]. The necessary qualities of the decomposition, and in particular the properties of the resulting diagonal matrix $\underline{\mathbf{S}}(z)$, which are required for this application, are then briefly discussed. A simple example is then given to illustrate how both of the methods perform as decomposition techniques, demonstrating a clear advantage of using the PSVD by PQRD algorithm over the previously proposed SBR2 approach. Most significantly, the PSVD by

PQRD algorithm enables the user to specify how small the coefficients associated with the off-diagonal polynomial elements must be driven for convergence. This is something that cannot be achieved when the PSVD is calculated using the SBR2 algorithm. Furthermore, the relative error of the decomposition, the computational time taken to calculate the decomposition and the orders of the final matrices are generally smaller when the decomposition is formulated using the PSVD by PQRD algorithm.

7.2 Technique 1: PSVD by PQRD

Given the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$, the objective of the PSVD by PQRD algorithm is to compute the polynomial paraunitary matrices $\underline{\mathbf{U}}(z) \in \underline{\mathbb{C}}^{p\times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q\times q}$ such that

$$\underline{\mathbf{U}}(z)\underline{\mathbf{A}}(z)\tilde{\underline{\mathbf{V}}}(z) \cong \underline{\mathbf{S}}(z) \tag{7.3}$$

where $\underline{\mathbf{S}}(z)$ denotes a diagonal polynomial matrix. As with the SBR2 algorithm and the PQRD techniques, it is often not possible to achieve exact diagonalisation of the polynomial matrix $\underline{\mathbf{A}}(z)$, as each element is now a polynomial with an associated set of coefficients, hence the reason for the approximate equality in equation (7.3). However, as this chapter will confirm a good approximation can be achieved. Note that the matrix to be decomposed, $\underline{\mathbf{A}}(z)$, need not be square, but it is assumed that it has at least as many rows as columns, i.e. $p \geq q$, and hence is generally referred to as a tall polynomial matrix. This algorithm could operate on a matrix with fewer rows than columns, but this would present an underdetermined problem, which is not the focus of this thesis.

The PSVD by PQRD algorithm operates as an iterative process where at each iteration two paraunitary matrices are formulated using one of the PQRD algorithms discussed in Chapter 5. Any of the PQRD algorithms can be used to compute the decomposition. Of course it is preferable to use the algorithm, which generally takes fewer iterations to converge to an upper triangular matrix and the one that obtains an upper triangular polynomial matrix with the smallest order. Note that these two properties of the decomposition are generally related. Over a series of iterations, the PSVD by PQRD algorithm transforms

the polynomial matrix $\underline{\mathbf{A}}(z)$ into an approximately diagonal matrix by applying a series of paraunitary matrices obtained using the PQRD.

7.2.1 The PSVD by PQRD Algorithm

The algorithm begins by calculating the paraunitary matrix $\underline{\mathbf{U}}_1(z) \in \underline{\mathbb{C}}^{p\times p}$. This matrix is calculated iteratively by formulating the PQRD of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ such that

$$\underline{\mathbf{U}}_1(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{A}}^{'}(z) \tag{7.4}$$

where $\underline{\mathbf{A}}^{'}(z) \in \underline{\mathbb{C}}^{p\times q}$ is an approximately upper triangular polynomial matrix, such that the coefficients of the polynomial elements beneath the diagonal satisfy

$$\left|a^{'}_{jk}(t)\right| < \epsilon \tag{7.5}$$

for $j = 2,\ldots,p$, $k = 1,\ldots,\min\{j-1,q\}$ and $\forall t \in \mathbb{Z}$, where $\epsilon > 0$ is a prespecified small value. Once $\underline{\mathbf{A}}^{'}(z)$ satisfies this stopping condition, set

$$\underline{\mathbf{A}}^{''}(z) = \underline{\widetilde{\mathbf{A}}}^{'}(z). \tag{7.6}$$

This polynomial matrix will be approximately lower triangular and will form the input to the subsequent step of the algorithm. Note that the diagonal polynomial elements of the matrix $\underline{\mathbf{A}}^{'}(z)$ will remain on the diagonal following the transformation and, in particular, the zero-lag diagonal coefficients will satisfy $a^{''}_{jj}(t) = (a^{'}_{jj}(-t))^*$ for $j = 1,\ldots,q$ and $\forall t \in \mathbb{Z}$ where $(\cdot)^*$ denotes the operation of complex conjugation of the polynomial coefficient. Note that in particular the diagonal zero-lag coefficient $a^{'}_{jj}(0)$ will remain in the same position following the transformation. This point is important for convergence of the algorithm.

Subsequently, the PQRD of the lower triangular polynomial matrix $\underline{\mathbf{A}}^{''}(z) \in \underline{\mathbb{C}}^{q\times p}$ is computed and so the paraunitary polynomial matrix $\underline{\mathbf{V}}_1(z) \in \underline{\mathbb{C}}^{q\times q}$ is calculated such that

$$\underline{\mathbf{V}}_1(z)\underline{\mathbf{A}}^{''}(z) = \underline{\mathbf{A}}^{'''}(z) \tag{7.7}$$

where the polynomial matrix $\underline{\mathbf{A}}^{'''}(z) \in \mathbb{C}^{q \times p}$ is approximately upper triangular and all coefficients of the polynomial elements beneath the diagonal are sufficiently small and therefore less than ϵ in magnitude, as in equation (7.5). This completes the first iteration of the PSVD by PQRD algorithm and the complete decomposition performed following this iteration is of the form

$$\underline{\mathbf{U}}_1(z)\underline{\mathbf{A}}(z)\underline{\tilde{\mathbf{V}}}_1(z) = \underline{\tilde{\mathbf{A}}}^{'''}(z) \tag{7.8}$$

where both of the paraunitary matrices $\underline{\mathbf{U}}_1(z)$ and $\underline{\mathbf{V}}_1(z)$ have been calculated as a series of elementary delay matrices interspersed by elementary rotation matrices using one of the algorithms for calculating the PQRD. This iterative process is repeated replacing $\underline{\mathbf{A}}(z)$ with $\underline{\tilde{\mathbf{A}}}^{'''}(z)$ until all coefficients of the off-diagonal polynomial elements of the matrix are sufficiently small, which is achieved when the following stopping condition is satisfied

$$|a_{jk}(t)| < \epsilon \tag{7.9}$$

for $j = 1, \ldots, p$, $k = 1, \ldots, q$ where $j \neq k$ and $\forall t \in \mathbb{Z}$. The value of the convergence parameter ϵ will be the same as that used for the stopping condition of both of the PQRDs calculated within each iteration of the decomposition to enable convergence of the algorithm. The algorithm is guaranteed to converge in this respect and the proof of convergence can be found in Section 7.2.3. Furthermore, a concise description of the PSVD by PQRD algorithm can be found in Appendix B.

Assuming the algorithm has converged following N iterations, the decomposition performed is of the form

$$\underline{\mathbf{U}}(z)\underline{\mathbf{A}}(z)\underline{\tilde{\mathbf{V}}}(z) = \underline{\mathbf{S}}_N(z) \tag{7.10}$$

where

$$\underline{\mathbf{U}}(z) = \underline{\mathbf{U}}_N(z) \ldots \underline{\mathbf{U}}_1(z), \tag{7.11}$$

$$\underline{\tilde{\mathbf{V}}}(z) = \underline{\tilde{\mathbf{V}}}_1(z) \ldots \underline{\tilde{\mathbf{V}}}_N(z) \tag{7.12}$$

and $\underline{\mathbf{S}}_N(z)$ is the approximately diagonal polynomial matrix resulting from N iterations of the PSVD by PQRD algorithm. The matrices $\underline{\mathbf{U}}_i(z)$ and $\underline{\mathbf{V}}_i(z)$ in the above expression denote

the pair of paraunitary matrices computed in the i^{th} iteration of the algorithm. The matrices $\underline{\mathbf{U}}(z) \in \underline{\mathbb{C}}^{p\times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q\times q}$ will therefore be paraunitary by construction.

Note that in the degenerate case where the input matrix is of order zero, i.e. the input is a matrix with scalar entries, then the decomposition will simply reduce to the conventional SVD of a matrix where the paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ will reduce to unitary matrices. Although the SVD of a scalar matrix can be calculated using this algorithm, it would be computationally more expensive than other techniques and as a consequence slower to implement.

7.2.2 Implementation of the Algorithm

The polynomial matrix truncation method suitable for non para-Hermitian polynomial matrices from Section 4.3.2, can also be used within the algorithm for calculating the PQRD. This ensures that the orders of the polynomial matrices do not grow unnecessarily large and as a result helps to minimise the computational load of the algorithm.

The stopping criterion of the algorithm, demonstrated in equation (7.9), could alternatively have been specified in terms of the proportion of the squared Frobenius norm of the matrix positioned in the off-diagonal elements, i.e. stop the algorithm when

$$\frac{\sum_t \sum_{j=1}^{p} \sum_{\substack{k=1 \\ k\neq j}}^{q} |a_{jk}(t)|^2}{\|\underline{\mathbf{A}}(z)\|_F^2} < \epsilon \tag{7.13}$$

where again $\epsilon > 0$ is a prespecified small value. This approach may be useful for some applications of the decomposition, but for the results demonstrated in this chapter the stopping condition of equation (7.9) is sufficient.

7.2.3 Convergence of the Algorithm

The proof of convergence is outlined assuming the PQRD-BC algorithm is used to calculate the PQRD at each step of the PSVD by PQRD algorithm, as this algorithm is often the fastest to converge for the examples used in this thesis. Note that this proof can easily be adapted

to apply to either the PQRD-BS or PQRD-SBR algorithms. To demonstrate convergence of the algorithm, the following measures are defined

$$N_4 = \sum_{\tau} \sum_{j=1}^{p} \sum_{k=1}^{q} |a_{jk}(\tau)|^2, \tag{7.14}$$

$$E_{Ck} = \sum_{\tau} \sum_{j=1}^{p} |a_{jk}(\tau)|^2, \tag{7.15}$$

$$E_{Rj} = \sum_{\tau} \sum_{k=1}^{q} |a_{jk}(\tau)|^2 \tag{7.16}$$

and

$$E_{ij}(\tau) = |a_{ij}(\tau)|^2, \tag{7.17}$$

which define the squared Frobenius norm of the matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$, the squared Frobenius norm of the k^{th} column of $\underline{\mathbf{A}}(z)$, the squared Frobenius norm of the j^{th} row of $\underline{\mathbf{A}}(z)$ and the magnitude squared of the polynomial coefficient $a_{jk}(\tau)$ respectively. Similarly, define the measures $\left(N_4', E_{Ck}', E_{Rj}', E_{ij}'(\tau)\right)$ for $\underline{\mathbf{A}}'(z)$, $\left(N_4'', E_{Ck}'', E_{Rj}'', E_{ij}''(\tau)\right)$ for $\underline{\mathbf{A}}''(z)$ and $\left(N_4''', E_{Ck}''', E_{Rj}''', E_{ij}'''(\tau)\right)$ for $\underline{\mathbf{A}}'''(z)$. The proof of convergence will initially be outlined assuming no truncation method is used and so the quantity N_4 will remain constant throughout all iterations of the decomposition.

The first step of the algorithm is to calculate the PQRD of the polynomial matrix $\underline{\mathbf{A}}(z)$. The resulting polynomial matrix from this decomposition, $\underline{\mathbf{A}}'(z)$, is guaranteed to converge to an upper triangular polynomial matrix, according to the stopping condition demonstrated in equation (7.5), by the proof for convergence of the appropriate PQRD algorithm. Note that the measure E_{Ck} is invariant to the application of the PQRD-BC algorithm over all columns of the matrix and so $E_{Ck}' = E_{Ck}$ for $k = 1, \ldots, q$. This is due to rotation matrices applied from the left of a matrix can only ever redistribute the squared Frobenius norm between elements in a column. Furthermore, the delay matrix will also keep energy constant within columns throughout the calculation of the PQRD.

Subsequently, the paraconjugate of this matrix is calculated to form $\underline{\mathbf{A}}''(z)$, which will be an approximately lower triangular polynomial matrix, where the magnitude of each coefficient

of each of the polynomial elements above the diagonal will be less than ϵ. The relationship between the coefficients of the two matrices can be expressed as $a''_{jk}(t) = \left(a'_{kj}(-t)\right)^*$ for $j = 1, \ldots, q$, $k = 1, \ldots, p$ and $\forall t \in \mathbb{Z}$, where $(\cdot)^*$ denotes the complex conjugation of each of the polynomial coefficients. In particular, note the following observations about this transformation:

1. $E''_{C'k} = E'_{Rk}$ for $k = 1, \ldots, q$.

2. All coefficients of the polynomial elements beneath the diagonal of a column will move into a column positioned to the right of its initial position and will be positioned above the diagonal.

3. All coefficients of each polynomial element on the diagonal of $\underline{\mathbf{A}}'(z)$ will remain in the same diagonal element following the transformation, but may have changed lag index.

4. All coefficients of the polynomial elements above the diagonal will move into a column positioned to the left of its initial position and will now be situated beneath the diagonal.

This final point is important for convergence, but first the final step of each iteration of the algorithm must be discussed.

The final step of the algorithm is to calculate the PQRD of the lower triangular polynomial matrix $\underline{\mathbf{A}}''(z)$ to obtain the upper triangular polynomial matrix $\underline{\mathbf{A}}'''(z)$. Following this transformation the magnitude of each coefficient of each polynomial element beneath the diagonal of $\underline{\mathbf{A}}''(z)$ will be driven less than ϵ by applying the appropriate elementary delay and rotation matrices within the PQRD-BC algorithm. During this transformation, the squared Frobenius norm of all of the coefficients of the polynomial elements positioned beneath the diagonal of $\underline{\mathbf{A}}''(z)$ will have been redistributed, as a result of driving these coefficients sufficiently small. Furthermore, the squared Frobenius norm of each column of the matrix will be invariant to the transformation performed, i.e. $E'''_{Ck} = E''_{Ck}$ for $k = 1, \ldots, q$, and so the squared Frobenius norm of the coefficients on and above the diagonal in each column will increase to account for the reduction below the diagonal.

Over this first iteration of the PSVD by PQRD algorithm, the coefficients of the polynomial elements, which were positioned above the diagonal of the k^{th} column, say, of $\underline{\mathbf{A}}'(z)$ and had magnitude greater than ϵ, will have firstly been moved into the k^{th} row of $\underline{\mathbf{A}}''(z)$. These coefficients will now be positioned in one of the columns to the left of k (i.e. in one of the columns $1, \ldots, k-1$) of the k^{th} row depending on their initial row index in $\underline{\mathbf{A}}'(z)$. When these coefficients are driven sufficiently small, such that each has magnitude less than ϵ, their excess energy will end up in either the diagonal element or any of the elements positioned above the diagonal within the same column of $\underline{\mathbf{A}}'''(z)$. This process is repeated, resulting in the squared magnitude of any coefficients larger than ϵ in polynomial elements above the diagonal of the matrix at the start of one iteration of the algorithm, being positioned on or above the diagonal of a column positioned to the left of its initial position at the end of the iteration. This process will continue through all iterations of the algorithm and so there will always be a movement of energy leftwards through the columns of the matrix until all coefficients are sufficiently small, i.e. less than ϵ.

Furthermore, the quantity $E_{11}(0)$ will increase monotonically throughout all iterations of the algorithm. It will increase as a consequence of driving any coefficient of any polynomial element beneath the diagonal of the first column to zero and is never affected by any rotations applied to zero coefficients in columns positioned to the right of it. In addition, this quantity will never be affected by the application of elementary delay matrices throughout any part of the decomposition as the coefficient $a_{11}(0)$ will remain in the same position throughout all iterations of the PSVD by PQRD algorithm. Now, this cannot continue indefinitely as the energy in the matrix is bounded from above by the initial value of N_4. Therefore, a point must be reached whereby all coefficients beneath the diagonal are less than ϵ by analogy with the proof of convergence of the SBR2 algorithm [7].

Finally, note that it will be possible for energy to move rightwards through the matrix as the coefficients beneath the diagonal of the matrix have only been driven sufficiently small according to a stopping condition and are not exactly equal to zero. Obviously, coefficients previously driven sufficiently small, which are now positioned in the area above the diagonal of the matrix, could increase in magnitude through a subsequent application of any of the PQRD

algorithms. However, at the next step of the algorithm any coefficients of any polynomial elements positioned above the diagonal will be moved into columns positioned to the left of their initial position, where they will then be systematically driven to zero according to the ordering within the PQRD-BC algorithm.

If the polynomial matrices are not truncated throughout the algorithm, then the quantity N_4 will remain constant throughout all iterations of the decomposition. However, this is not practical for most applications of the algorithm and so a truncation method will be implemented forcing the measure N_4 to decrease according to the truncation method described in Section 4.3.2. However, although this quantity can decrease it will still constitute an upper bound on $E_{11}(0)$ and so convergence of the algorithm is guaranteed. This proof of convergence is also easily extended for the two remaining algorithms for calculating the QR decomposition of a polynomial matrix discussed in Chapter 5.

7.3 Technique 2: PSVD by PEVD

This method is an already existing technique for calculating the SVD of a polynomial matrix and operates by using the SBR2 algorithm to calculate the PEVD of two polynomial matrices. Suppose we wish to calculate the PSVD of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ such that

$$\underline{\mathbf{A}}(z) = \underline{\tilde{\mathbf{U}}}(z)\underline{\mathbf{S}}(z)\underline{\mathbf{V}}(z) \tag{7.18}$$

where $\underline{\tilde{\mathbf{U}}}(z) \in \underline{\mathbb{C}}^{p\times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q\times q}$ are both paraunitary matrices and $\underline{\mathbf{S}}(z) \in \underline{\mathbb{C}}^{p\times q}$ denotes a diagonal matrix. Then using the polynomial matrix, $\underline{\mathbf{A}}(z)$, two para-Hermitian polynomial matrices, $\underline{\mathbf{A}}(z)\underline{\tilde{\mathbf{A}}}(z) \in \underline{\mathbb{C}}^{p\times p}$ and $\underline{\tilde{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{q\times q}$, can be generated. These matrices could alternatively be expressed in terms of the PSVD shown in equation (7.18) as

$$\begin{aligned}\underline{\mathbf{A}}(z)\underline{\tilde{\mathbf{A}}}(z) &= \underline{\tilde{\mathbf{U}}}(z)\underline{\mathbf{S}}(z)\underline{\mathbf{V}}(z)\underbrace{\tilde{\mathbf{V}}(z)}_{\mathbf{I}}\tilde{\mathbf{S}}(z)\underline{\mathbf{U}}(z) &(7.19)\\ &= \underline{\tilde{\mathbf{U}}}(z)\underline{\mathbf{S}}(z)\tilde{\mathbf{S}}(z)\underline{\mathbf{U}}(z) &(7.20)\end{aligned}$$

and

$$\tilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) = \tilde{\underline{\mathbf{V}}}(z)\tilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{U}}(z)\tilde{\underline{\mathbf{U}}}(z)\underline{\mathbf{S}}(z)\underline{\mathbf{V}}(z) \quad (7.21)$$

$$= \tilde{\underline{\mathbf{V}}}(z)\tilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{S}}(z)\underline{\mathbf{V}}(z) \quad (7.22)$$

where the matrices $\underline{\mathbf{S}}(z)\tilde{\underline{\mathbf{S}}}(z) \in \mathbb{C}^{p\times p}$ and $\tilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{S}}(z) \in \mathbb{C}^{q\times q}$ are both diagonal. Equations (7.20) and (7.22) both constitute the PEVD of the matrices $\underline{\mathbf{A}}(z)\tilde{\underline{\mathbf{A}}}(z)$ and $\tilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$. The paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ can therefore be calculated by applying the SBR2 algorithm to the para-Hermitian matrices $\underline{\mathbf{A}}(z)\tilde{\underline{\mathbf{A}}}(z)$ and $\tilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$ in turn. As each element of a polynomial matrix represents an FIR filter, exact diagonalisation is not always possible, but as Chapter 3 has demonstrated a good approximation is achievable. Therefore, the SBR2 algorithm when applied to the para-Hermitian matrices will generate the approximations of the diagonal matrices $\underline{\mathbf{S}}(z)\tilde{\underline{\mathbf{S}}}(z)$ and $\tilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{S}}(z)$ according to a stopping condition, such as that demonstrated by equation (3.25).

Finally, once the two PEVDs have been calculated to obtain the paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$, the diagonalised matrix $\underline{\mathbf{S}}(z)$ is calculated according to equation (7.1). Note that if a truncation method is used when formulating the two decompositions in equations (7.20) and (7.22), then the decomposition will not be exactly energy preserving and so equation (7.2) will not hold precisely. However, with a suitable choice of the truncation parameter μ, a good level of decomposition can be achieved.

The PSVD of the polynomial matrix $\underline{\mathbf{A}}(z)$ has then been obtained allowing the matrix to be decomposed as demonstrated by equation (7.1). Note that the two PEVDs demonstrated by equations (7.20) and (7.22) are both guaranteed to converge to an approximately diagonal matrix, by the proof of convergence for the SBR2 algorithm outlined in Section 3.4.

As with the previous PSVD technique, if the matrix $\underline{\mathbf{A}}(z)$ is of order zero and so each element is a scalar, this decomposition technique will reduce to the conventional SVD, where $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ are unitary matrices. However, this technique would not be the method of choice for the decomposition as computationally it is more expensive.

There is a problem with using this technique to form the PSVD; it is impossible to have

any direct control over how small the off-diagonal coefficients of the matrix $\underline{\mathbf{S}}(z)$ will be following the decomposition. Although the maximum magnitude of the off-diagonal coefficients of the diagonal matrices obtained by the SBR2 algorithm, i.e. $\underline{\mathbf{S}}(z)\widetilde{\underline{\mathbf{S}}}(z)$ and $\widetilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{S}}(z)$, can be specified, this will not allow strict control over the maximum magnitude of the off-diagonal coefficients of $\underline{\mathbf{S}}(z)$. For example, applying the SBR2 algorithm to $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$, the coefficients of the off-diagonal polynomial elements of the resulting diagonalised matrix $\underline{\mathbf{S}}(z)\widetilde{\underline{\mathbf{S}}}(z)$ must be less than a prespecified small value $\epsilon > 0$ and so the coefficients must satisfy

$$\left|\left[\underline{\mathbf{S}}(z)\widetilde{\underline{\mathbf{S}}}(z)\right]_{jkt}\right| = \left|\sum_{l=1}^{q}\sum_{t_1}\sum_{\substack{t_2 \\ t_2-t_1=t}} s_{jl}(t_1)s_{kl}^{*}(t_2)\right| < \epsilon \tag{7.23}$$

where $j,k = 1,\ldots,p$ for $j \neq k$ and $\forall t \in \mathbb{Z}$. However, this does not give any insight to the magnitude of the coefficients of the off-diagonal polynomial elements of $\underline{\mathbf{S}}(z)$. Therefore the maximum possible size of the coefficients of the off-diagonal polynomial elements of the diagonalised matrix $\underline{\mathbf{S}}(z)$ cannot be specified in advance, resulting in little control over how diagonal the resulting matrix will be. Clearly, for the application of the decomposition, where a strictly diagonal matrix may be required for channel equalisation, this could affect the error rate performance. This is discussed further in Chapter 8 of this thesis. Note that this is not a problem in the degenerate case where the decomposition of a scalar matrix is required, as exact diagonalisation of the matrices $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$ and $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$ is possible.

7.4 Uniqueness of Solutions

Following from Section 3.8, it can be seen that the two paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ obtained from formulating the PSVD of a polynomial matrix are not unique. Note that the PSVD algorithms presented in this thesis generate only an approximate PSVD and so the solutions of the two techniques will not be the same. For the potential applications of the decomposition discussed in Chapter 8, non-uniqueness of the paraunitary matrices does not present any problem. Before demonstrating the two decomposition methods, it is important to briefly highlight the application of the decomposition in signal processing and in particular

stress the properties and qualities required of the decomposition for this application. A full discussion of the application of the decomposition can be found in Chapter 8.

7.5 Requirements of the PSVD for Applications

The polynomial singular value decomposition can be used in multiple antenna channel decomposition problems to transform a MIMO channel into a set of orthogonal SISO channels, where a maximum likelihood decoder can then be applied to each SISO channel to obtain an estimate of one of the source signals. However, if the order of the diagonalised matrix $\underline{\mathbf{S}}(z)$ is too large then equalisation of the channel becomes difficult. Generally, filters of length greater than 20 may be considered too large. For this reason, the order of the resulting diagonalised channel matrix $\underline{\mathbf{S}}(z)$ is very important.

Obviously, it may also be preferable to use the method of decomposition which is less computationally expensive and therefore quicker to run and so this is another factor of the decomposition that must be examined.

Finally, it is important to look at the relative error of the decomposition to ensure that the accuracy of the decomposition has not been compromised when implementing either of the PSVD techniques. Unlike the relative error used for the previous decompositions, two different relative errors are proposed for the PSVD. The first relative error is defined as

$$E_1^{rel} = \frac{\left\| \underline{\mathbf{A}}(z) - \underline{\widetilde{\mathbf{U}}}(z)\underline{\mathbf{S}}(z)\underline{\mathbf{V}}(z) \right\|_F}{\|\underline{\mathbf{A}}(z)\|_F} \tag{7.24}$$

and will demonstrate the amount of error due to truncating the polynomial matrices throughout all iterations or steps of the decomposition methods. However, for application purposes of the decomposition, a strictly diagonal polynomial matrix $\underline{\mathbf{S}}(z)$ is required, i.e. all off-diagonal elements must equal zero. The two PSVD techniques proposed in this chapter only generate an approximately diagonal polynomial matrix and so the second relative error measure is defined as

$$E_2^{rel} = \frac{\left\| \underline{\mathbf{A}}(z) - \underline{\widetilde{\mathbf{U}}}(z)\underline{\widehat{\mathbf{S}}}(z)\underline{\mathbf{V}}(z) \right\|_F}{\|\underline{\mathbf{A}}(z)\|_F}, \tag{7.25}$$

where $\widehat{\underline{\mathbf{S}}}(z)$ is equal to the diagonal matrix $\underline{\mathbf{S}}(z)$ with all off-diagonal coefficients set equal to zero. This measure is a much superior measure of how well the decomposition technique has performed, as it will not only account for the error due to truncating the polynomial matrices, but will also assess how well the algorithm has diagonalised the polynomial matrix.

7.6 Numerical Example

To illustrate the two different methods for calculating the PSVD, each is applied in turn to the polynomial matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{4\times3\times5}$, which was previously used for the example in Section 5.8. The polynomial elements of this matrix had complex coefficients, where both the real and imaginary parts were drawn randomly from a standardised Gaussian probability density function. A graphical representation for this polynomial matrix was given in Figure 5.2.

7.6.1 PSVD by PQRD

For this example, the PQRD-BS algorithm is used within the PSVD by PQRD algorithm, as it required the least number of iterations to converge. The truncation method suitable for non para-Hermitian polynomial matrices from Section 4.3.2 was implemented, with the truncation parameter set as $\mu = 10^{-4}$. The PSVD by PQRD algorithm ran for 10 iterations, until the magnitude of each off-diagonal coefficient of $\underline{\mathbf{S}}(z)$ was less than 10^{-2}. This required a total of 984 PQRD iterations (i.e. EPGRs) over all 10 iterations of the PSVD by PQRD algorithm. Note that the number of PQRD iterations per iteration of the PSVD by PQRD algorithm decreases as the algorithm progresses. In fact, the majority of the 984 EPGRs were applied in the first few iterations of the PSVD by PQRD algorithm, with the tenth consisting of only six EPGRs. The paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ obtained from the decomposition are illustrated in Figures 7.1 and 7.2 respectively. The diagonal matrix $\underline{\mathbf{S}}(z)$ found using the PSVD by PQRD algorithm can be seen in Figures 7.3.

The magnitude of the largest off-diagonal coefficient of $\underline{\mathbf{S}}(z)$ was found to be 9.3×10^{-3}. The Frobenius norm of the coefficients of the off-diagonal polynomial elements of this approximately diagonal matrix was calculated as 0.04, which accounts for 0.37% of the total

Frobenius norm of the matrix. The Frobenius norm of $\underline{\mathbf{S}}(z)$ did decrease from 9.86 to 9.83 from truncating the polynomial matrices, giving a relative error of $E_1^{rel} = 0.1084$. Furthermore, assuming all the coefficients of the off-diagonal polynomial elements of the matrix are equal zero gave a second relative error of $E_2^{rel} = 0.1086$, showing that a good level of matrix diagonalisation has been achieved.

From inspection of the diagonal matrix $\underline{\mathbf{S}}(z)$ demonstrated in Figure 7.3, the diagonal polynomial elements are not all centred about the same lags in each element. However, a series of elementary delay matrices can be applied to the polynomial matrix $\underline{\mathbf{S}}(z)$ to realign the series of coefficients of each diagonal element to be centred about the same lag. For example, the coefficient with maximum magnitude in each diagonal polynomial element could have a shift applied to it so that it becomes the coefficient of z^0. Note that one of the paraunitary matrices from the decomposition must also be altered accordingly. This procedure will help concentrate the majority of the Frobenius norm of the matrix into a smaller number of lags, whilst maintaining the paraunitary transformation. As a result of this realignment it is possible to further reduce the order of the diagonal matrix by applying the truncation function once more to the realigned matrix, using the same value of the truncation parameter μ. Applying a final series of elementary delay matrices to the diagonal matrix $\underline{\mathbf{S}}(z)$ shown in Figure 7.3, the order of this matrix was reduced from 22 to 17 whilst maintaining the same value of relative error. The results obtained from applying the PSVD by PQRD method to the polynomial matrix $\underline{\mathbf{A}}(z)$ can be seen in Table 7.1.

7.6.2 PSVD by PEVD

To obtain the PSVD of $\underline{\mathbf{A}}(z)$ using the second method, the SBR2 algorithm was applied to the polynomial matrices $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$ and $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$ in turn to obtain the paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$. Each implementation of the SBR2 algorithm ran until the magnitude of each off-diagonal coefficient of the resulting diagonalised polynomial matrices $\underline{\mathbf{S}}(z)\widetilde{\underline{\mathbf{S}}}(z)$ and $\widetilde{\underline{\mathbf{S}}}(z)\underline{\mathbf{S}}(z)$ fell below 10^{-2}, which required a total of 362 iterations over both applications of the SBR2 algorithm. Note that the two truncation functions from Sections 4.3.1 and 4.3.2 were used throughout both implementations of the SBR2 algorithm with $\mu = 10^{-4}$, which

allowed at most, this proportion of the squared Frobenius norm of the matrix to be lost each time a matrix is truncated. This stopped the order of the polynomial matrices within the algorithm growing unnecessarily large and as a results also prevented the algorithm from being excessively slow to implement.

The approximately diagonal polynomial matrix $\underline{\mathbf{S}}(z)$, is then calculated using the two parauntiary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ according to equation (7.1). This matrix is illustrated in Figure 7.4 and is order of 41, considerably larger than the order of the diagonal matrix found using the PSVD by PQRD algorithm. Furthermore, the magnitude of the largest coefficient associated with an off-diagonal polynomial element of $\underline{\mathbf{S}}(z)$ was found to be 0.29. Again considerably larger than the same measure found using the PSVD by PQRD algorithm, which was found to be 9.3×10^{-3}. The two paraunitary polynomial matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ generated from this method are demonstrated graphically in Figures 7.5 and 7.6.

The Frobenius norm of the resulting diagonal polynomial matrix, $\underline{\mathbf{S}}(z)$, obtained from the PSVD by PEVD routine was found to be 9.85, which means only 0.16% of the Frobenius norm of the input matrix $\underline{\mathbf{A}}(z)$ has been lost through truncating the polynomial matrices. The relative error E_1^{rel} was found to be 0.0684, which is less than that observed with the decomposition performed by the PSVD by PQRD algorithm, however this did not take into account that the matrix $\underline{\mathbf{S}}(z)$ is only approximately diagonal. The Frobenius norm of the coefficients of the off-diagonal polynomial elements of this matrix was found to be 1.52 accounting for 1.57% of the total Frobenius norm of the matrix $\underline{\mathbf{S}}(z)$. The total relative error for the decomposition was found to be $E_2^{rel} = 0.1670$ and this measure is considerably larger than that found using the PQRD approach. From inspection of the diagonal matrix $\underline{\mathbf{S}}(z)$ in Figure 7.4, the diagonal polynomial elements are not all centred around the same lag. As with the results from the PSVD by PQRD method, aligning the diagonal elements of $\underline{\mathbf{S}}(z)$ so that the coefficient with maximum magnitude for each polynomial is realigned as the coefficient of z^0 and then truncating the matrix once more using the same value for the truncation parameter μ, the order of the matrix can be reduced to 36. However, this is still considerably larger than the order of the diagonal matrix obtained using the PSVD by PQRD algorithm. The results obtained from applying this decomposition technique to the polynomial matrix $\underline{\mathbf{A}}(z)$

are contained in Table 7.1. This table demonstrate that the PQRD method outperforms the SBR2 approach for a number of reasons; it obtains a lower relative error, the resulting matrix $\underline{\mathbf{S}}(z)$ is also more diagonal and the order of this matrix is typically shorter.

	PSVD By	
	SBR2	**PQRD**
Magnitude of largest off-diagonal polynomial coefficient	0.2871	9.3×10^{-3}
Frobenius norm of off-diagonal polynomial elements of $\underline{\mathbf{S}}(z)$	1.5244	0.0363
Order of $\underline{\mathbf{S}}(z)$	41	22
Order of $\underline{\mathbf{U}}(z)$	36	47
Order of $\underline{\mathbf{V}}(z)$	24	36
E_1^{rel}	0.0684	0.1084
E_2^{rel}	0.1670	0.1086
After Final Delay		
Order of $\underline{\mathbf{S}}(z)$	36	17
Order of $\underline{\mathbf{U}}(z)$	27	47
E_1^{rel}	0.0687	0.1084
E_2^{rel}	0.1670	0.1086

Table 7.1: Results obtained from applying the two methods for calculating the PSVD to the polynomial matrix $\underline{\mathbf{A}}(z)$ where the truncation parameter is set as $\mu = 10^{-4}$ and the stopping criterion as $\epsilon = 10^{-2}$ in both methods.

For the results demonstrated so far it has not been entirely fair to compare the two techniques as they did not achieve the same level of decomposition. For example, it would be better if both decompositions had obtained similar values for the Frobenius norm of the off-diagonal elements or a similar magnitude of the largest off-diagonal coefficient. The off-diagonal elements of the diagonal matrix $\underline{\mathbf{S}}(z)$ could have been driven smaller by setting a tighter convergence bound when applying the SBR2 algorithm to the polynomial matrices $\underline{\mathbf{A}}(z)\underline{\tilde{\mathbf{A}}}(z)$ and $\underline{\tilde{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$. For example, if the coefficients of these matrices were required to be less than 10^{-7}, the magnitude of the largest off-diagonal coefficient of $\underline{\mathbf{S}}(z)$ is now found

to be 1.7×10^{-2}, but this took 5708 SBR2 iterations and the order of the diagonal matrix is now 163 (although this can be reduced to 150 if the diagonal elements are aligned around the zero lag of the matrix). Note that to achieve the tighter convergence bound, the truncation parameter must also be reduced and so, for the results above, $\mu = 10^{-8}$. The order of the diagonal matrix could be further reduced to 19 by truncating the final realigned diagonal matrix again setting $\mu = 0.01$. The relative error of the decomposition is now $E_2^{rel} = 0.11$.

The magnitude of the largest coefficient of an off-diagonal polynomial element is still considerably larger than the 9.3×10^{-3} found using the PSVD by PQRD algorithm. Moreover, the order of the resulting diagonal matrix, which was found to be 17 using the PQRD approach, is also slightly larger using the SBR2 method. The relative error for the decomposition,which was found to be $E_2^{rel} = 0.11$ using the PQRD method, is the same, although the PSVD by PQRD algorithm was also considerably faster to run for these results, taking only 0.44 seconds. The SBR2 approach took 22.57 seconds[1]. The comparative results for the two algorithms are contained in Table 7.2.

	PSVD By	
	SBR2	**PQRD**
Magnitude of largest off-diagonal polynomial coefficient	1.7×10^{-2}	9.3×10^{-3}
Order of $\underline{\mathbf{S}}(z)$	19	17
E_2^{rel}	0.11	0.11
Value of ϵ	10^{-7}	10^{-2}
Value of μ	10^{-8}	10^{-4}
Computational Time (Seconds)	22.57	0.44
Number of Iterations	5708	984

Table 7.2: Observed results from the two decomposition techniques when both achieve approximately the same level of PSVD, allowing a much fairer comparison of the two algorithms.

[1]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

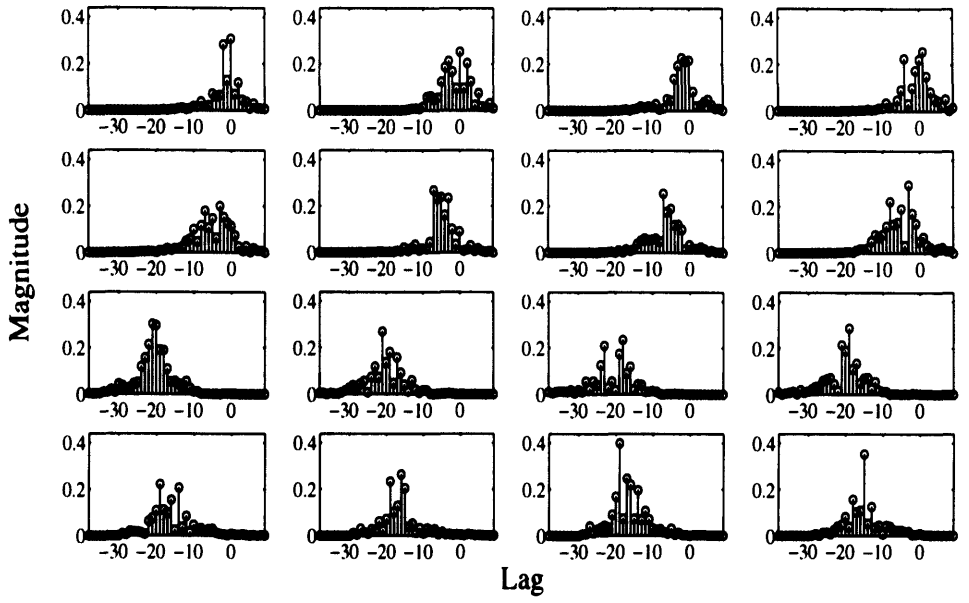

Figure 7.1: A stem plot representation of the paraunitary matrix $\underline{\mathbf{U}}(z)$ obtained from the PSVD by PQRD algorithm.

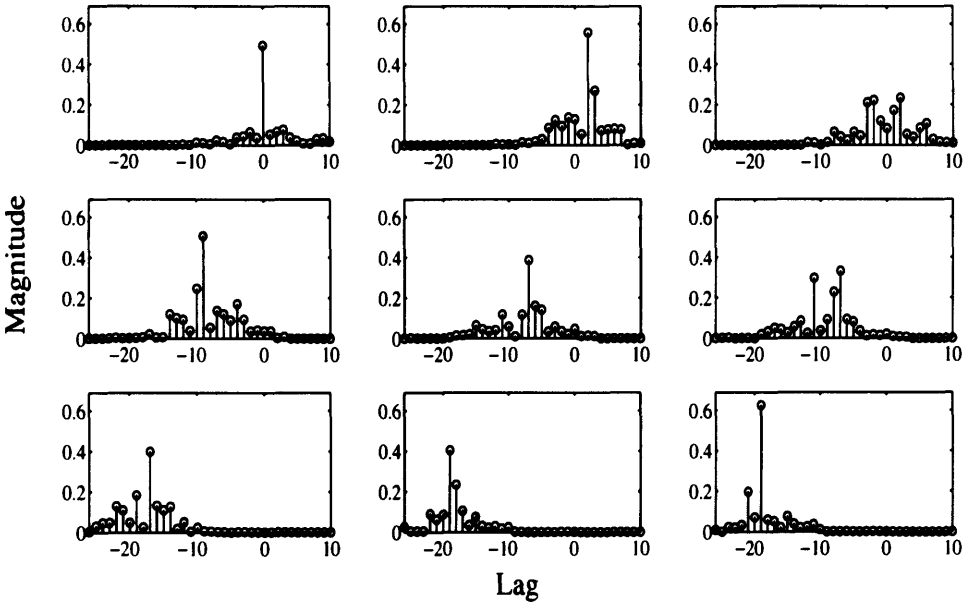

Figure 7.2: A stem plot representation of the paraunitary matrix $\underline{\mathbf{V}}(z)$ obtained from the PSVD by PQRD algorithm.

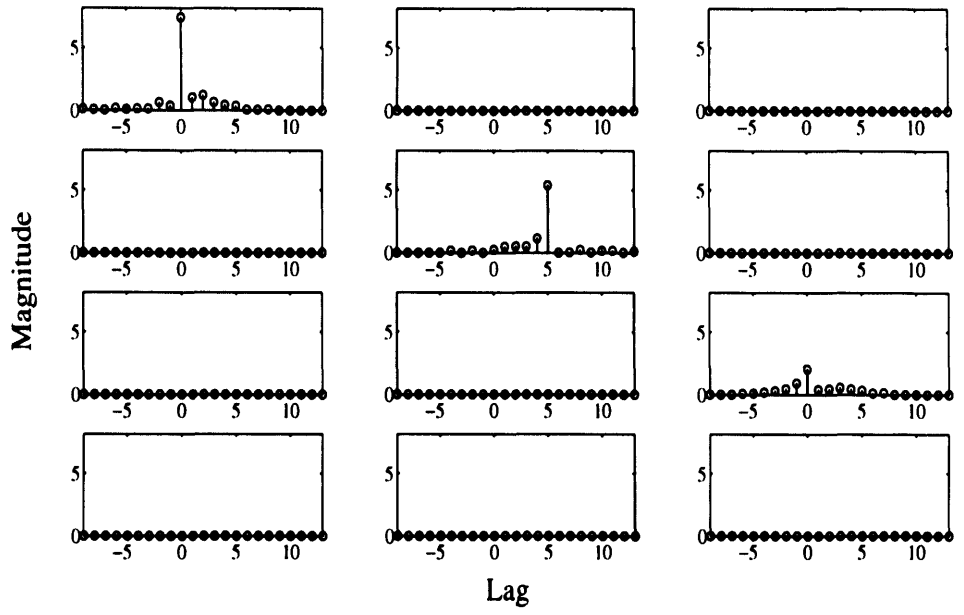

Figure 7.3: A stem plot representation of the diagonal matrix $\underline{\mathbf{S}}(z)$ obtained when the PSVD by PQRD algorithm is applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ shown in Figure 5.2.

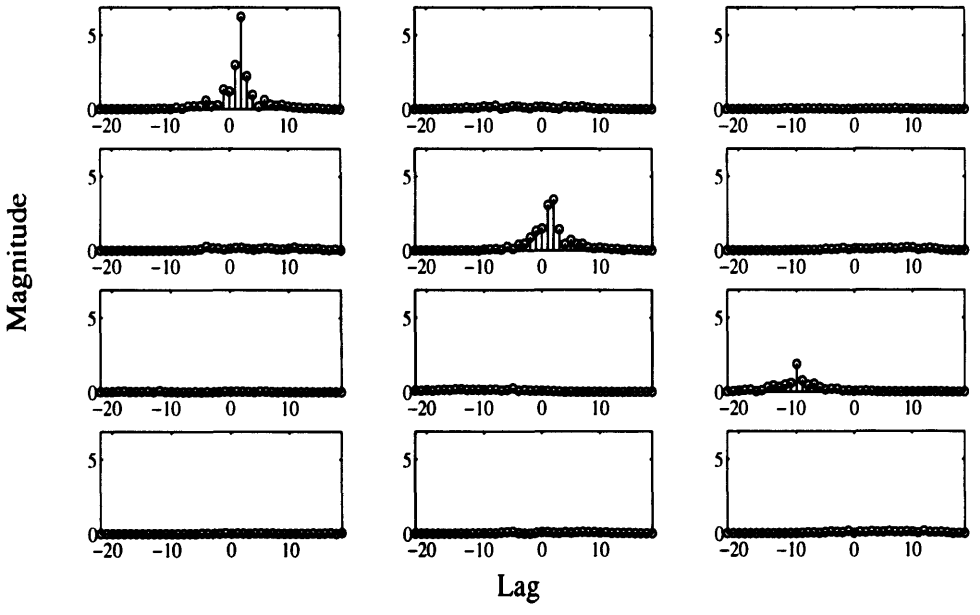

Figure 7.4: A stem plot representation of the diagonal matrix $\underline{\mathbf{S}}(z)$ obtained when the PSVD by SBR2 technique is applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ shown in Figure 5.2.

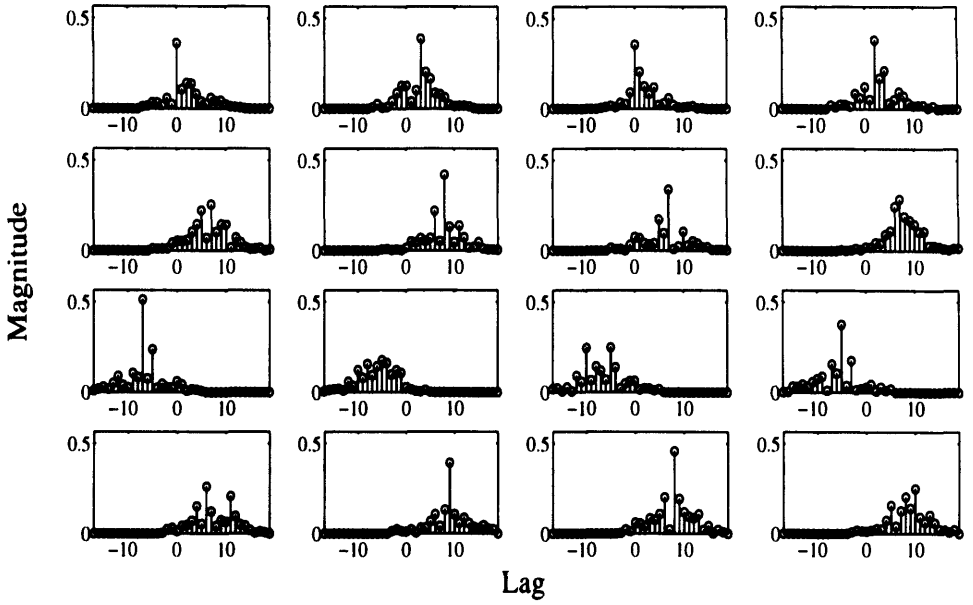

Figure 7.5: A stem plot representation of the paraunitary matrix $\underline{\mathbf{U}}(z)$ obtained from the PSVD by SBR2 technique.

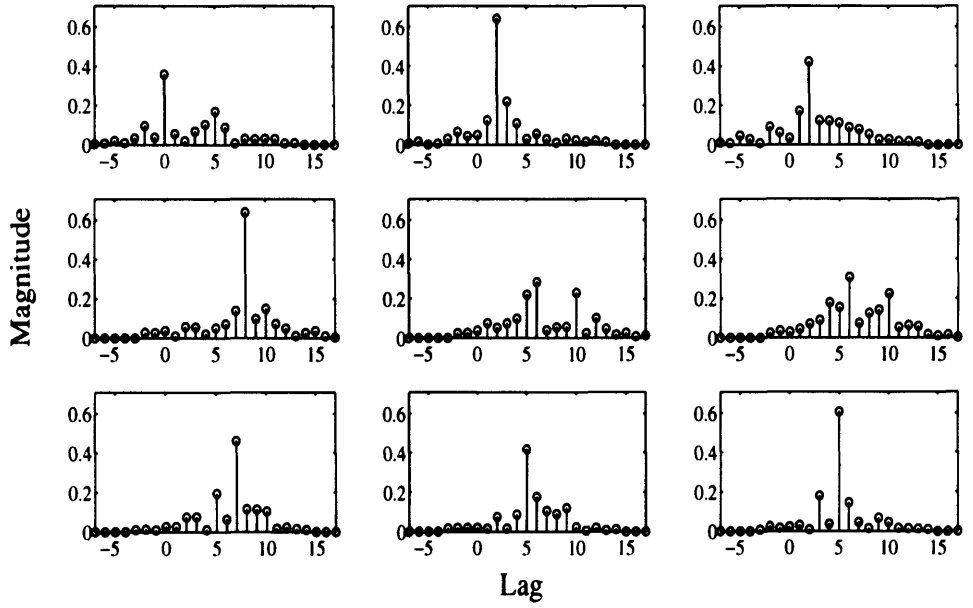

Figure 7.6: A stem plot representation of the paraunitary matrix $\underline{\mathbf{V}}(z)$ obtained from the PSVD by SBR2 technique.

7.7 Computational Complexity of the Polynomial SVD Methods

It is difficult to compare the two decomposition techniques in terms of their computational complexity, as the computational complexity of one iteration of either the SBR2 or PQRD algorithm is entirely dependent on the dimension and order of the polynomial matrix at that iteration. If the SBR2 and the PQRD-BS algorithms are both applied to a polynomial matrix of the same dimensions, then the SBR2 algorithm will be slightly less computationally expensive to run, despite the fact that the SBR2 algorithm will apply the elementary delay and rotation matrices from both the left and the right, whereas with the PQRD they are only applied from the left. This is due to the SBR2 algorithm exploiting the para-Hermitian structure of the polynomial matrix $\underline{\mathbf{A}}(z)$ and only storing just over half of the polynomial elements of the matrix. However, to calculate the PSVD of a matrix the two algorithms are not applied to the same polynomial matrix and so the computational load at each step is difficult to compare. In the example of the previous section, the PQRD algorithm is applied to the polynomial matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{4\times3\times4}$, whereas the SBR2 algorithm is applied to the polynomial matrices $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z) \in \mathbb{C}^{4\times4\times8}$ and $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) \in \mathbb{C}^{3\times3\times8}$. The SBR2 algorithm is applied to polynomial matrices with more off-diagonal polynomial coefficients.

7.8 Conclusions

A new method for calculating the singular value decomposition of a polynomial matrix $\underline{\mathbf{A}}(z)$ has been presented. The technique operates by iteratively calculating the PQRD of a polynomial matrix to transform it into a diagonal polynomial matrix and is therefore referred to as the PSVD by PQRD algorithm. The algorithm has been compared to a previously proposed method for calculating the PSVD, which operates using the SBR2 algorithm to calculate the PEVD of the matrices $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$ and $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$ to generate the left and right hand singular vectors respectively. The PSVD by PQRD algorithm has a couple of clear advantages over the PSVD by PEVD method and for the simple numerical example presented in this chapter,

it clearly outperforms the PSVD by PEVD technique.

The main advantage of using the PQRD method over the SBR2 approach is that the user can specify how small the coefficients of the off-diagonal polynomial elements of the matrix are to be driven before starting the algorithm. It is impossible to do this using the existing PSVD method, as the SBR2 algorithm is not directly applied to the polynomial matrix whose PSVD is being calculated. The only way this can be achieved, is with a considerable amount of trial and error to find the appropriate values of μ and ϵ, required to drive the off-diagonal coefficients sufficiently small.

Secondly, the PQRD approach was found to be computationally much faster (approximately 50 times faster) to obtain a similar level of decomposition, when both techniques were applied to the same polynomial matrix $\underline{\mathbf{A}}(z) \in \mathbb{C}^{4\times3\times4}$. Note that the time taken by the SBR2 approach did not even include the time taken to find the appropriate choice of the truncation parameter μ and stopping criterion ϵ.

Finally, the order of the diagonal matrix obtained using the PSVD by PQRD algorithm is generally considerably less than that found with using the SBR2 approach. This is an important advantage of the decomposition when it is applied to MIMO communication problems as discussed in Chapter 8.

When using either of the two methods, it is important to align the diagonal polynomial elements of the resulting diagonal matrix, so that the majority of the Frobenius norm of the matrix is centred around the same lags of the matrix. As the example in this chapter has shown, this can help reduce the order of the diagonal matrix, at hardly any additional computational cost or gain in the error of the decomposition.

Note when using the truncation functions within the PSVD by PEVD method, it is better to be cautious and set a very small value of the truncation parameter throughout, for example $\mu = 10^{-8}$, and then truncate the final matrix at the very end of the decomposition to reduce the order to a suitable value for the application purposes of the decomposition. More examples of this decomposition technique can be found in Chapter 8, where the potential applications of the decomposition are examined.

Chapter 8

Applications of the Polynomial Matrix Decompositions

8.1 Introduction

Chapter two described how polynomial matrices are widely used in the context of DSP and, in particular, to communication systems, where they are used to describe a convolutive channel and therefore describe a MIMO system. This chapter examines the potential applications of each of the three polynomial matrix decompositions discussed in this thesis (the PEVD, the PQRD and the PSVD) in this context. Table 8.1 details the possible applications of the decompositions and refers the reader to the appropriate section in the thesis where the application is discussed.

Finally, some simulated results are presented to further demonstrate the capabilities, but also the potential applications, of the two algorithms introduced in this thesis (i.e. the algorithms for calculating the PQRD and the PSVD) to MIMO channel equalisation problems. However, before detailing these applications some background to MIMO communication systems must be given.

Application	Decomposition	Section
Strong Decorrelation	PEVD	3.6.1
MIMO Channel Equalisation	PQRD	8.4.1
MIMO Channel Exploitation	PSVD	8.5.1
Optimal Paraunitary Filter Bank Design	PSVD	8.5.4

Table 8.1: The potential applications of the different polynomial matrix decompositions and the location of a discussion of this application within the thesis.

8.2 MIMO Communication Systems

Both the PQRD and the PSVD algorithms introduced in this thesis have potential applications to MIMO communication systems. A brief description of this type of communication system can be found in Section 2.4.3 and a basic noise free baseband digital communication system [25] can be seen in Figure 8.2, where $\underline{\mathbf{s}}(z)$ denotes the signals to be transmitted, $\underline{\mathbf{x}}(z)$ the received signals and $\underline{\widehat{\mathbf{s}}}(z)$ the estimated source signals.

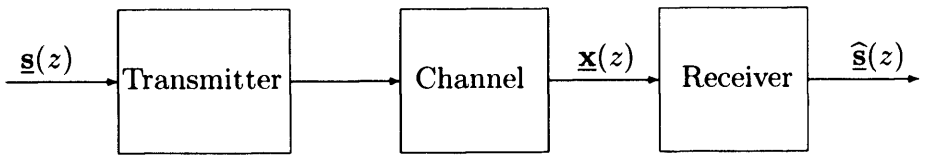

Figure 8.1: Block diagram for a basic noise free baseband communication system.

It is assumed that the channel will be convolutive for the results in this thesis and so mathematically the mixing process of sending the data from q antennas to be received at p sensors can be expressed by means of the convolutive mixing model given in equation (2.28). The complete process, from transmitter to receiver, is known as a MIMO communication system [49, 60, 61].

The aim of the transmitter is to ensure the message is in a suitable form for transmission,

whilst the receiver aims to operate on the received signal to obtain an estimate of the original message. However, this is not a simple task due a number of factors that can affect the transmission of data through a channel and hence lead to distorted signals. These factors are

1. **Intersymbol interference (ISI)** caused by the dispersion in the channel.
2. **Thermal noise**, which is present at the receiver.
3. With MIMO channels, there is also the problem of **co-channel interference (CCI)**.

The task of removing the effect of these factors from the received signal has been the subject of much research in the field of DSP. There are now several methods for achieving this, such as using a process of orthogonal frequency-division mulitplexing (OFDM) with a cyclic prefix [2] or linear or non-linear interference cancellers [62]. There is currently a lot of interest and research being undertaken in these fields, especially with the recent advancements in wireless technology [2, 62]. Problems in this field include the limited availability of radio frequency spectrum and a complex time-varying wireless environment [2]. Also, there is now a demand for higher data rates, higher network capacity and matching wireline link reliability.

For the discussion and results presented in this chapter, only a simple communication system is considered. The received signals from the MIMO system will be distorted due to the effects of ISI, CCI and the noise present at the receiver. It is assumed that the polynomial channel matrix $\underline{\mathbf{C}}(z)$ from equation (2.30) has been previously estimated by passing a training sequence through the channel. Some algorithms exist that directly equalise the channels given only the received signals; this is known as blind channel equalisation. However, these algorithms can be slow to converge, are subject to local minima and are generally unsuitable for wireless channels [63]. As a result, it has been suggested that it is better to first identify the channel and then perform the equalisation. The two new polynomial matrix decompositions proposed in this thesis, the PQRD and the PSVD, can be used to help solve this problem. Assuming that the polynomial channel matrix for the system is known, then either the PQRD or the PSVD can be used as part of this system to mitigate the CCI and transform the problem into a set of SISO problems, i.e. a set of problems involving only one transmitter and one receiver. The ISI from each of the single channel problems can then be removed by

equalisation. Several methods of equalising a SISO communication channel are now discussed. Note that problems will be encountered with this method if the channel matrix for the system is rank deficient and this aspect will be discussed individually for each application.

8.2.1 Channel Equalisation

Channel equalisation is the process used to remove the effect of ISI by modelling the channel inverse [62] and is used in many communication devices such as modems and digital televisions. If the channel is minimum phase, then it has a linear inverse model and so a linear equalisation solution exists. The various techniques for performing channel equalisation can be split into the following four classes.

1. **Linear Equalisers (LE)**. These equalisers use an inverse filter to compensate for the variation in the frequency response. This method is simple, but is not very effective with channels that have deep fades.

2. **Decision Feedback Equalisers (DFE)**. These attempt to reconstruct ISI from past symbol decisions. Again this method is simple, but unlike LEs this method will have the potential for error propagation.

3. **Maximum Likelihood Sequence Estimation (MLSE)**. These methods find the most likely sequence of symbols given the received signal and the possible options. These techniques are robust, but can be computationally complex. The Viterbi algorithm, which is discussed in Section 8.2.2, can be used for MLSE equalisation.

4. **Turbo Equalisers (TE)** This is an iterative code based on the maximum a posteriori probability (MAP) criterion. This type of equaliser can significantly reduce the SNR penalty caused by ISI [64], but its computational complexity is higher than the MLSE scheme and it introduces a delay (latency) in processing.

The main problems with the different techniques for equalisation are that they are computationally expensive, have problems tracking time-varying channels and can only produce

sequences of outputs with a significant time delay. For the results presented in this chapter, an MLSE based on the Viterbi algorithm is implemented to demonstrate the potential applications of the polynomial matrix decompositions.

8.2.2 The Viterbi Algorithm

The Viterbi algorithm is a non-sequential decoding algorithm, motivated by the work undertaken by A.J. Viterbi in the 1960's on behalf of NASA to improve the efficiency of their space communication system. The algorithm was first proposed in [65] as a maximum likelihood decision device, which can be applied to any sequence of symbols that can be modelled as a Markov chain, [66, 67]. As many observed phenomena can be modelled as Markov processes, the algorithm has a vast range of applications. For example the algorithm is currently used in all mobile telephone systems and is incorporated in satellite digital TV receivers and mobile phone handsets. The primary application of the Viterbi algorithm is the maximum likelihood decoding of convolutionally coded digital signals transmitted over noisy channels, making the algorithm applicable to channel equalisation problems. However, the algorithm now has a much broader range of applications and can be applied to problems in areas such as speech recognition [68] and DNA sequence analysis [69].

The Viterbi algorithm operates by attempting to determine the most likely sequence of symbols, given all the possible options, i.e. it attempts to calculate the most probable path through a Markov graph [66]. The Viterbi algorithm is therefore a computationally efficient method for removing ISI present from the received signal of a SISO communication system. The solution to the MLSE problem will generally be close to optimal, however, there is a problem using this technique. The complexity of the MLSE will increase exponentially with channel order and so is not a suitable technique to use if the environment has a large delay spread [2]. For example, if M defines the size of the symbol alphabet and N defines the number of interfering symbols contributing to the ISI, then the Viterbi algorithm must calculate M^{N+1} metrics for each new received signal [62]. For this reason, the resulting order of the transformed polynomial matrix when using either the PQRD or PSVD algorithms as a preprocessing step to channel equalisation, is very important. However, for the results

presented in this chapter, the maximum order of the transformed channel matrix (following the polynomial matrix decomposition) is limited to ensure that this equalisation scheme can be used. Note that this is acceptable for the results presented in this chapter, as limiting the orders of the matrices did not significantly compromise the relative error. This point is discussed further for each of the worked examples. Furthermore, the aim of this chapter is to illustrate the ability of the polynomial matrix decompositions as a method for transforming a polynomial MIMO channel matrix into a set of SISO channels and so the method of channel equalisation is not the focus of the research. For practical applications, a suboptimal channel equalisation method, which is computationally less expensive, could be used if required.

8.3 Performance Measures

8.3.1 Relative Error

As with the previous chapters, the two relative errors for the polynomial matrix decompositions can be calculated as

$$E^{rel} = \frac{\left\| \underline{\mathbf{A}}(z) - \underline{\widehat{\mathbf{A}}}(z) \right\|_F}{\| \underline{\mathbf{A}}(z) \|_F} \tag{8.1}$$

where if the measure is calculated for the PQRD then

$$\underline{\widehat{\mathbf{A}}}(z) = \underline{\widetilde{\mathbf{Q}}}(z)\underline{\widehat{\mathbf{R}}}(z) \tag{8.2}$$

where $\underline{\widehat{\mathbf{R}}}(z)$ is equal to the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ with all coefficients beneath the diagonal set equal to zero. If calculating the relative error of the PSVD, then this matrix is given by

$$\underline{\widehat{\mathbf{A}}}(z) = \underline{\widetilde{\mathbf{U}}}(z)\underline{\widehat{\mathbf{S}}}(z)\underline{\mathbf{V}}(z) \tag{8.3}$$

where $\underline{\widehat{\mathbf{S}}}(z)$ is equal to the diagonal matrix $\underline{\mathbf{S}}(z)$ with all off-diagonal coefficients set equal to zero. This measure signifies how much information is lost in the associated polynomial matrix decompositions either by truncating the polynomial matrices or from the requirement of a strictly diagonal or upper triangular polynomial matrix for the applications of the

decomposition.

8.3.2 Average Bit Error Rates

The average bit error rate (BER) of a sequence defines the ratio of the number of characters incorrectly received to the total number transmitted during a specified time interval. Firstly, define the error metric

$$e_t = \begin{cases} 0 & \text{if } \mathbf{s}(t) = \hat{\mathbf{s}}(t) \\ 1 & \text{if } \mathbf{s}(t) \neq \hat{\mathbf{s}}(t) \end{cases} \tag{8.4}$$

where $\mathbf{s}(t)$ defines the true source signal at time t and $\hat{\mathbf{s}}(t)$ the estimated, where $t = 0, \ldots, T-1$. The average BER is then calculated as

$$BER = \sum_{t=0}^{T-1} \frac{e_t}{T}. \tag{8.5}$$

This measure can be used to describe the functionality of a digital communications system and therefore has been calculated for both the PQRD and PSVD applications described in this chapter.

8.4 Potential Application of the PQRD

One possible application of the PQRD is in MIMO communications where it is often necessary to reconstruct a set of signals, which have been transmitted through a convolutive channel, using only the received signals and an estimate of the polynomial channel matrix. In this situation, the data will have been distorted due to both the effects of co-channel interference (CCI) and multi-path propagation of the transmitted signals, which can then result in inter-symbol interference (ISI). The problem of reconstructing the data sequence from the convolutively mixed received data is termed as MIMO channel equalisation. For this application, it is assumed that the channel matrix has previously been estimated. This can be achieved by passing a training sequence through the system, however, it is beyond the scope of this thesis to discuss methods for estimating the channel matrix.

8.4.1 MIMO Channel Equalisation

This potential application of the decomposition is first discussed for the simpler narrowband case, where scalar matrices are observed and so the conventional methods of calculating the QRD of a scalar matrix can be used [2,6].

Narrowband (Scalar Matrix) Case

If a set of signals is instantaneously mixed, then the relative delay between signals can be modelled as a phase shift and so a matrix of complex scalar entries is sufficient to describe the mixing. This will mean that there is no ISI present, however, there is still CCI and the conventional QRD for scalar matrices can be used to remove this. In this situation, if the channel matrix $\mathbf{C} \in \mathbb{C}^{p \times q}$ has been previously estimated, its QRD can be formulated, using a conventional technique such as Givens rotations or Householder reflections [6]. Once this decomposition has been calculated, the upper triangular structure of the transformed matrix can be exploited, allowing the set of source signals to be easily determined from the received signals, using back substitution.

For this application, it is required that $p \geq q$ and that the channel matrix for the system is of full column rank to enable the complete set of source signals to be determined. If the channel matrix is rank deficient, then a number of diagonal elements of the upper triangular matrix will equal zero. Unfortunately, this will mean that it is impossible to obtain estimates of the source signal with the same row index as the zero element(s) in the diagonal matrix. Furthermore, it will not be possible to estimate the remaining sources with row indices less than this value. This point has been discussed in Example 6.2.6, where the PQRD of a rank deficient polynomial matrix has been calculated.

Broadband (Polynomial Matrix) Case

This technique is easily extended to broadband signal processing, where polynomial channel matrices are now observed. However, for this case the conventional techniques for calculating the QRD of a scalar matrix cannot be used to determine this decomposition, as each element of the channel matrix will now be a polynomial with an associated set of coefficients. However,

provided the channel matrix has been estimated, its QRD can be calculated according to one of the PQRD algorithms detailed in Chapter 5, thus allowing the channel matrix to be transformed into an approximately upper triangular polynomial matrix by means of a paraunitary transformation. This then enables the MIMO channel equalisation problem to be transformed, using back substitution, into a set of SISO channel equalisation problems, which can each be solved using a MLSE based on the Viterbi algorithm [62, 66, 67]. This process will now be explained.

It is assumed that a set of source signals $\mathbf{s}(t) \in \mathbb{C}^{q\times 1}$, where $t \in \{0,\ldots,T-1\}$, are emitted from q independent sources through a convolutive channel to be received at an array of p sensors, where it is assumed that $p \geq q$. The mixing model for the set of convolutively mixed received signals $\mathbf{x}(t) \in \mathbb{C}^{p\times 1}$ can be expressed as

$$\underline{\mathbf{x}}(z) = \underline{\mathbf{C}}(z)\underline{\mathbf{s}}(z) + \underline{\mathbf{n}}(z) \tag{8.6}$$

where $\underline{\mathbf{C}}(z) \in \underline{\mathbb{C}}^{p\times q}$ denotes the polynomial mixing (or channel) matrix and $\underline{\mathbf{x}}(z)$, $\underline{\mathbf{s}}(z)$ and $\underline{\mathbf{n}}(z)$ denote algebraic power series, i.e. a series of the form $\underline{\mathbf{x}}(z) = \sum_{t=-\infty}^{\infty} \mathbf{x}(t)z^{-t}$, of the received signals, the source signals and the noise process, which has variance $\sigma^2\mathbf{I}_p$. In digital communications, the source signals $\mathbf{s}(t)$ are generally drawn from a finite constellation, such arise in binary or quaternary phase-shift keying (BPSK/QPSK).

The first step to achieve MIMO channel equalisation is to calculate the QR decomposition of the polynomial channel matrix $\underline{\mathbf{C}}(z)$ using any of the three algorithms detailed in Chapter 5, such that

$$\underline{\mathbf{C}}(z) = \underline{\mathbf{Q}}(z)\underline{\mathbf{R}}(z), \tag{8.7}$$

where $\underline{\mathbf{Q}}(z) \in \underline{\mathbb{C}}^{p\times p}$ denotes the polynomial paraunitary transformation matrix and $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{p\times q}$ is an approximately upper triangular polynomial matrix. The convolutive mixing model of equation (8.7) can then be rewritten as

$$\underline{\mathbf{x}}'(z) = \underline{\mathbf{R}}(z)\underline{\mathbf{s}}(z) + \underline{\mathbf{n}}'(z) \tag{8.8}$$

where $\underline{\mathbf{x}}'(z) = \underline{\tilde{\mathbf{Q}}}(z)\underline{\mathbf{x}}(z)$ and $\underline{\mathbf{n}}'(z) = \underline{\tilde{\mathbf{Q}}}(z)\underline{\mathbf{n}}(z)$. As the polynomial matrix $\underline{\mathbf{Q}}(z)$ is paraunitary and its application is a linear transformation, $\underline{\mathbf{n}}'(z)$ is also a Gaussian noise process with an identical norm. Note that to enable the set of p equations demonstrated in vector form by equation (8.8) to become a set of q single channel equalisation problems, all elements beneath the diagonal of $\underline{\mathbf{R}}(z)$, which are approximately equal to zero, are assumed to be equal to zero. This will affect the accuracy of the decomposition and possibly the error rate performance of the method for MIMO channel equalisation, however no numerical problems have been encountered when applying the algorithm to a wide range of polynomial matrices. Furthermore, if the relative error of the decomposition is too large, it can be reduced by decreasing the value of the stopping criterion ϵ.

Now provided the channel matrix is of full column rank, the MIMO channel equalisation problem can be transformed into a set of q single channel equalisation problems using back substitution. Beginning with the q^{th} element of $\underline{\mathbf{x}}'(z)$ from equation (8.8), this can be expressed as

$$\underline{x}_q'(z) = \underline{r}_{qq}(z)\underline{s}_q(z) + \underline{n}_q'(z), \tag{8.9}$$

which is a single channel equalisation problem. This can now be solved, to obtain an estimate of the q^{th} source signal, $\hat{s}_q(t)$, using an MLSE based on the Viterbi decoder [62,66,67]. Once this estimate has been attained, it can now be used to formulate a single channel equalisation problem involving $\underline{s}_{q-1}(z)$ as follows

$$\underline{x}_{q-1}'(z) - \underline{r}_{(q-1)q}(z)\underline{s}_q(z) = \underline{r}_{(q-1)(q-1)}(z)\underline{s}_{q-1}(z) + \underline{n}_{q-1}'(z), \tag{8.10}$$

which can again be solved using an MLSE. Furthermore, once the estimates $\hat{s}_{i+1}(t), \ldots, \hat{s}_q(t)$ have been calculated, the i^{th} single channel equalisation problem can be formulated as

$$\underline{x}_i'(z) - \sum_{j=i+1}^{q} \underline{r}_{ij}(z)\underline{s}_j(z) = \underline{r}_{ii}(z)\underline{s}_i(z) + \underline{n}_i'(z). \tag{8.11}$$

Provided the set of signals are estimated according to the ordering $i = q, q-1, \ldots, 1$, each equation then reduces to a single channel equalisation problem. Each SISO equalisation

problem can be solved to obtain an estimate of the i^{th} source signal, $\hat{s}_i(t)$, using the previously estimated sources $\hat{s}_j(t)$ for $j = i+1, \ldots, q$. However, as with the narrowband or scalar matrix case, if the channel matrix of the system is not of full column rank the system will be underdetermined and there will be fewer equations than unknowns. The PQRD of the channel matrix could still be calculated, but it will not be possible to perform the process of back substitution followed by equalisation to obtain an estimate of every source signal.

The role of the back substitution, which is made possible by calculating the PQRD, is to remove CCI. The second step of applying the MLSE, enables the ISI observed in each single channel equalisation problem to be eliminated. However, this second step can only be achieved once the transmitted signal, which is to be estimated, is expressed in terms of a SISO system and so the two steps must operate together, with back substitution enabling another SISO problem to be solved. The mitigation of CCI will enable better frequency reuse within a communications system and as a result will improve the network spectrum efficiency [2]. A block diagram of the proposed baseband communication system using the PQRD can be seen in Figure 8.2.

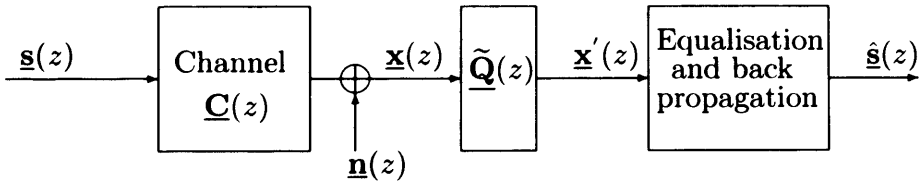

Figure 8.2: Block diagram for a basic baseband communication system using the PQRD.

8.4.2 Filter At the Transmitter

Alternatively, the PQRD of the paraconjugate of the channel matrix $\underline{\mathbf{C}}(z) \in \underline{\mathbb{C}}^{p \times q}$ could have been calculated such that

$$\widetilde{\underline{\mathbf{C}}}(z) = \underline{\mathbf{Q}}(z)\underline{\mathbf{R}}(z) \tag{8.12}$$

where $\underline{\mathbf{Q}}(z) \in \underline{\mathbb{C}}^{q \times q}$ is the polynomial paraunitary transformation matrix and $\underline{\mathbf{R}}(z) \in \underline{\mathbb{C}}^{q \times p}$

denotes the upper triangular polynomial matrix. The convolutive mixing model of equation (8.6) could then have been rewritten as

$$\underline{\mathbf{x}}(z) = \underline{\tilde{\mathbf{R}}}(z)\underline{\mathbf{s}}'(z) + \underline{\mathbf{n}}(z) \tag{8.13}$$

where $\underline{\mathbf{s}}'(z) = \underline{\tilde{\mathbf{Q}}}(z)\underline{\mathbf{s}}(z)$ and $\underline{\tilde{\mathbf{R}}}(z) \in \underline{\mathbb{C}}^{p\times q}$ is now a lower triangular polynomial matrix. Again a two step process of back substitution and the application of a MLSE can be used to estimate the set of source signals. Using this decomposition, which for the scalar matrix case is often referred to as an RQ or LQ decomposition, allows the signals to be filtered at the transmitter rather than the receiver, which was demonstrated previously in Section 8.4.1. This will mean that the channel state information (CSI) is only required at either the receiver or transmitter. If this was to be used in mobile communications, for example, it would require no MIMO processing in the mobile, but only at the transmitter and could therefore be advantageous.

8.4.3 Numerical Examples

To illustrate this proposed application of the PQRD to MIMO communications, two channel matrices, $\underline{\mathbf{C}}_1(z) \in \mathbb{C}^{4\times3\times4}$ and $\underline{\mathbf{C}}_2(z) \in \mathbb{C}^{5\times5\times4}$ are generated. Both of the channel matrices are of full rank and have been chosen for their varying properties and different structures. However, before discussing these matrices, some issues about the implementation of the algorithms for calculating the PQRD are discussed.

Comments on Implementation

For proposed application of the PQRD to MIMO communications, any of the three algorithms introduced in Chapter 5 could be used to calculate the decomposition. However, it is obviously preferable to use the fastest and most accurate of the three algorithms. From the examples of Chapter 6, the PQRD-BC algorithm has consistently demonstrated the best performance, typically requiring the least number of EPGRs and computational time to converge. Furthermore, the orders of the resulting matrices were often shorter and the relative error of the decomposition less. However, these potential advantages of the algorithm are not

guaranteed for every polynomial matrix. Therefore, for the worked examples of this chapter, the decomposition has been formulated using each of the three algorithms and then the resulting matrices $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{Q}}(z)$ obtained from the algorithm providing the best performance have been used to perform the MIMO channel equalisation.

The order of the resulting polynomial matrix $\underline{\mathbf{R}}(z)$ is critical for this application due to the computational complexity of the equalisation step being directly proportional to the final order of this matrix. In particular, if a MLSE method based on the Viterbi algorithm is used to perform the equalisation, then the computational complexity of each step will be exponentially proportional to the order of $\underline{\mathbf{R}}(z)$. The worked examples in this chapter use this method and so the order of the upper triangular polynomial matrix formulated by the decomposition for each example has been restricted to 14.

For each implementation of each of the algorithms for calculating the PQRD, the truncation method suitable for non para-Hermitian polynomial matrices from Section 4.3.2 was applied to both of the polynomial matrices at the end of each iteration of each of the algorithms, with the truncation parameter μ set equal to 10^{-6}. This will optimise the speed of the algorithm, whilst ensuring that a fairly accurate polynomial matrix decomposition is formulated. However, if the order of the resulting upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ following the decomposition is larger than 14, then the matrix has been truncated using the fixed bound truncation method. Whenever truncating the order of any of the polynomial matrices, it is important to recalculate the relative error of the final polynomial matrix decomposition, to ensure that the accuracy has not been substantially compromised. Furthermore, the value of the stopping criterion ϵ, from equation (5.13), can also affect the relative error of the decomposition and if it is not sufficiently small then this will be reflected in the measure E^{rel}. For each example of this chapter the parameter was set equal to 10^{-3}.

It can also be beneficial to realign the zero-lag coefficients of the polynomial elements of the upper triangular polynomial matrix resulting from the PQRD as discussed in Chapter 6. This will help to concentrate the series of coefficients associated with each polynomial element around the set of zero-lag coefficients of the polynomial matrix. This will often allow the polynomial matrices to be further truncated at little, or often no, additional cost to the

accuracy of the decomposition performed.

Example 1

The first polynomial mixing matrix $\underline{\mathbf{C}}_1(z) \in \underline{\mathbb{C}}^{4\times 3}$ was generated to describe the propagation of three source signals on to five sensors. Each of the polynomial elements of the matrix was chosen to be a fourth order FIR filter, where both the real and imaginary parts were drawn from a uniform distribution in the range $[-1, 1]$. The matrix was also normalised so that $\|\underline{\mathbf{C}}_1(z)\|_F = 1$. A graphical representation of this polynomial matrix can be found in Appendix D.

Firstly, each of the three algorithms for calculating the PQRD introduced in Chapter 5 was applied to this polynomial channel matrix and the results obtained from each decomposition can be seen in Table 8.2, where g defines the magnitude of the dominant coefficient of the resulting approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$. The PQRD-BC algorithm demonstrated the best performance of the three algorithms, requiring only two sweeps of the algorithm and a total of 291 iterations over twelve steps of the algorithm to converge to a point where $g = 9.59 \times 10^{-4}$. This algorithm required significantly fewer iterations that either the PQRD-BS or the PQRD-SBR algorithms to converge to a point where the magnitude of every coefficient associated with the polynomial elements positioned beneath the diagonal of $\underline{\mathbf{R}}(z)$ is less than 10^{-3}. Furthermore, the orders of the resulting polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$ are smaller than those obtained using the other two algorithms and this algorithm also obtained the least relative error. This decomposition was therefore used to perform the MIMO channel equalisation of this channel matrix. Figures illustrating the series of coefficients for the two polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$ obtained using this algorithm can be seen in Appendix D.

Subsequently, three independent BPSK source signals, each of length 1000, were generated and convolutively mixed using the channel matrix $\underline{\mathbf{C}}_1(z)$ following the mixing model demonstrated by equation (2.28), where N defines the order of the polynomial channel matrix and for this example is equal to four. Gaussian noise representative of thermal noise, with spatial covariance $\sigma^2\mathbf{I}_4$, was then added to each of the receive sensors to give a desired

	PQRD-BS	PQRD-BC	PQRD-SBR
Number of Iterations	567	291	620
Order of $\underline{\mathbf{R}}(z)$	89	42	54
Order of $\underline{\mathbf{Q}}(z)$	98	46	61
g	9.97×10^{-4}	9.59×10^{-4}	9.83×10^{-4}
E^{rel}	1.52×10^{-2}	9.26×10^{-3}	1.11×10^{-2}
Computational Time (Seconds)	0.81	0.30	1.60

Table 8.2: The results obtained from applying the three algorithms for calculating the PQRD to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

signal-to-noise ratio at the receiver (RSNR). For this experiment the RSNR can be calculated as

$$\text{RSNR} = 10\log_{10}\left(\frac{Tr\left\{\underline{\mathbf{C}}_1(z)\widetilde{\underline{\mathbf{C}}}_1(z)\right\}\Big|_{t=0}}{p\,\sigma^2}\right) \tag{8.14}$$

where p defines the number of receivers, which for this example is equal four, and $|_{t=0}$ denotes the scalar matrix containing the coefficients of z^0 of the polynomial matrix. Through the two step process of back substitution and applying the MLSE described in 8.4.1, an estimate for each of the source signals was obtained. The average bit error rate (BER) for each of the estimated source signals was calculated, where the variance of the additive noise was chosen to give varying levels of SNR. This was carried out for 100 independent Monte-Carlo realisations, using the same channel matrix $\underline{\mathbf{C}}_1(z)$, but generating new source signals and noise terms for each realisation. The average of these results can be seen in Table 8.3, where for RSNR levels $10 - 20$ dB the technique offers excellent error rate performance.

For these results, the order of the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ was truncated using the fixed bound truncation method, reducing the order of the matrix from 42 to 14, which then enabled the MLSE to be implemented. As a result, the relative error of the decomposition increased from 9.26×10^{-3} as observed in Table 8.2 to 0.0493. Similar results have been presented in [70], however, these results were calculated using the PQRD-BS algorithm to formulate the PQRD of the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$. For these results a larger value for the truncation parameter was used ($\mu = 10^{-3}$) throughout the decompo-

sition, and as a result, the relative error of the decomposition was found to be considerably larger with $E^{rel} = 0.168$. For this reason, the error rate performance was slightly worse, demonstrating that it is better to truncate the orders of the polynomial matrices using the energy based truncation method throughout the algorithm using a very small value for the truncation parameter μ and then truncate the final order of the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ to be sufficiently small for the channel to be equalised.

	Average BER							
	$\underline{\mathbf{C}}_1(z)$			$\underline{\mathbf{C}}_2(z)$				
SNR (dB)	**Source 1**	**Source 2**	**Source 3**	**Source 1**	**Source 2**	**Source 3**	**Source 4**	**Source 5**
−5	0.1906	0.1893	0.2589	0.2362	0.2644	0.2965	0.2607	0.3604
0	0.0619	0.0639	0.1239	0.1287	0.1471	0.1942	0.1430	0.2696
5	0.0025	0.0033	0.0177	0.0319	0.0422	0.0744	0.0344	0.1532
10	0	0	0.0001	0.0012	0.0014	0.0051	0.0013	0.0389
15	0	0	0	0	0	0.0001	0	0.0014
20	0	0	0	0	0	0	0	0

Table 8.3: Average BERs for the estimated BPSK sources for the MIMO channel equalisation problem. The results are demonstrated for the two channel matrices $\underline{\mathbf{C}}_1(z) \in \mathbb{C}^{4\times3\times4}$ and $\underline{\mathbf{C}}_2(z) \in \mathbb{C}^{5\times5\times4}$, and for varying levels of RSNR.

Example 2

For the second example, the polynomial channel matrix $\underline{\mathbf{C}}_2(z) \in \mathbb{C}^{5\times5\times4}$ was generated such that each coefficient associated with each of the polynomial elements of the matrix is of the form $\frac{1}{2}(a + ib)$, where both a and b are drawn randomly from a Gaussian distribution with mean zero and unit variance. This matrix will correspond to a quasi-static block of a Rayleigh frequency selective channel. The matrix was normalised so that $\|\underline{\mathbf{C}}_2(z)\|_F = 1$. As with the first example, each of the three algorithms for calculating the PQRD was applied to this polynomial matrix, to assess their performance and therefore choose the most appropriate algorithm to formulate the PQRD of the channel matrix. The results from applying each of the three algorithms to this polynomial matrix are contained in Table 8.4.

Once again, the PQRD-BC algorithm demonstrated the best performance of the three algorithms, requiring significantly less iterations and therefore EPGRs to converge. This algorithm required only two sweeps of the algorithm and a total of 536 iterations to converge to a point where the magnitude of each coefficient associated with a polynomial element positioned beneath the diagonal of the polynomial matrix $\underline{\mathbf{R}}(z)$ is less than 10^{-3}. Note that the orders of the polynomial matrices generated by the algorithm are larger than those obtained using the other two algorithms. However, this is not an issue for this application of the decomposition, as the orders of the upper triangular polynomial matrices obtained from each of the three algorithms are too large for the equalisation step and must therefore be truncated further. Note that the relative error observed when using the PQRD-BC algorithm is slightly larger than that obtained using the PQRD-SBR algorithm, but the SBR approach required considerably more time to converge and has therefore not been used to perform the decomposition. Graphical representations of the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$ and the two polynomial matrices obtained when applying the PQRD-BC algorithm to this matrix can be seen in Appendix D.

	PQRD-BS	**PQRD-BC**	**PQRD-SBR**
Number of Iterations	750	536	856
Order of $\underline{\mathbf{R}}(z)$	64	74	63
Order of $\underline{\mathbf{Q}}(z)$	69	82	68
g	9.84×10^{-4}	9.99×10^{-4}	9.96×10^{-4}
E^{rel}	0.0161	0.0148	0.0129
Computational Time (Seconds)	1.25	0.86	3.50

Table 8.4: Results obtained from applying the three algorithms for calculating the PQRD to the polynomial matrix $\underline{\mathbf{C}}_2(z)$.

To enable equalisation of the channel using an MLSE based on the Viterbi algorithm, the order of the resulting approximately upper triangular polynomial matrix must be sufficiently small. For this reason, the order of the polynomial matrix $\underline{\mathbf{R}}(z)$ must be truncated to have an order of 14. However, to ensure that enforcing this will not compromise the accuracy of the decomposition unnecessarily, the series of coefficients associated with the polynomial

elements of the polynomial matrices obtained using the PQRD-BC algorithms are realigned by applying a series of final elementary delay matrices as discussed in Chapter 6. Applying a final alignment to this example, the order of $\underline{\mathbf{R}}(z)$ was reduced from 74 to 63 and the order of $\underline{\mathbf{Q}}(z)$ from 82 to 73, at no additional cost to the relative error of the decomposition. Furthermore, once the order of $\underline{\mathbf{R}}(z)$ has been reduced to 14 to enable the equalisation of the polynomial channel matrix, the relative error of the decomposition was calculated as 0.1168.

A set of five BPSK source signals, each of length 1000, was generated and then convolutively mixed according to equation (2.28), where N is again equal to four. Noise is then added to the convolutively mixed signals, where the variance of the noise has been chosen to give a desired RSNR value, which is calculated according to equation (8.14) where $p = 5$ and σ^2 defines the variance of the noise. Estimates of the five source signals are then calculated using the process of back substitution and equalisation detailed in Section 8.4.1 and their BER calculated. The RSNR for the experiment was allowed to vary from -5 to 20 dB in increments of five. The results, for each RSNR value were then averaged over 100 realisations. The same channel matrix was used throughout, however, the signals and noise terms were generated afresh for each realisation. The average BER results are contained in Table 8.3, which demonstrate this technique has shown excellent error-rate performance for all five sources for RSNR values in the range 10 to 20 dB.

8.4.4 Discussion of the Effect of Relative Error on Bit Error Rate

The performance of the equaliser during this process, will be affected by the relative error of the decomposition performed. However, the order of the polynomial matrix $\underline{\mathbf{R}}(z)$ must be sufficiently small to enable the equaliser to be applied and so, for most polynomial channel matrices, there will be some level of relative error encountered by truncating the order of this matrix. Other factors that will affect the relative error of the decomposition are the choice of the stopping criterion ϵ and the truncation parameter μ. Also the final truncation and realignment, if it is used, will affect this measure. Note that it is better to use a small value

of μ when calculating the decomposition and then truncate the order of the final polynomial matrix $\underline{\mathbf{R}}(z)$, if its order is still too large to implement the equalisation step. This will help minimise the relative error encountered. For the worked examples in this thesis, the choice of both parameters has always been determined to optimise the speed and accuracy of the algorithm used to calculate the PQRD.

Note that the transformation performed by the PQRD is norm-preserving in the columns of the polynomial matrix. However, truncating the polynomial matrices throughout the decomposition will mean that this is no longer true (although it will be approximately true using a suitable value for the truncation parameter μ) and each column of the resulting upper triangular matrix is affected by truncation by varying amounts. Due to the matrix being transformed into the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, truncation of this matrix throughout the PQRD algorithm will generally result in a more upper triangular polynomial matrix and hence, over the whole transformation, the most energy will typically be lost in the far right column of the matrix. The least energy will be lost in the first column, with all columns in between the first and last having increasing amounts of energy lost (due to the columns of the input matrix having more non-zero elements in them).

8.4.5 Conclusions

The numerical examples presented in this section have demonstrated that the algorithms for calculating the PQRD, introduced in this thesis, can be used successfully as a preprocessing step in MIMO communication systems to transform a problem of MIMO channel equalisation into a series of SISO channel problems, which can be solved using an equalisation scheme such as MLSE. Furthermore, the decomposition has shown to yield good error rate performance when applied to a quasi static channel with a constant power profile, which is the typical structure of a wireless channel. Future work could assess the performance of this method when using other equalisation techniques. Furthermore, pre- and post-processing techniques, such as interleaving and error correction coding can be used to improve the average BER performance performance of the system [71]. Similar results have been found using QPSK source signal, but are not presented in this thesis.

Note that it is better to set the value of the truncation parameter μ as small as possible, as this will mean a more accurate decomposition. If possible, it is obviously best to obtain the decomposition without truncating the polynomial matrices by setting $\mu = 0$, however, computationally the algorithm becomes very slow to implement. Instead, the polynomial matrices can be truncated throughout the algorithm using a very small value for μ at each iteration and then truncate the final polynomial matrices using the fixed bound truncation method described in Section 4.2 to ensure that they are of appropriate orders for the application.

Tests have demonstrated, but are not included in this thesis, that the relative error of the decomposition can be allowed to increase to approximately 0.1 for these examples without compromising the average BER results. Further research could be carried out to determine the amount of error that can be allowed within the decomposition without significantly affecting the error rate performance with this equalisation scheme. Note that the order of the approximately upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$ is critical to this application of the decomposition as the computational complexity of the MLSE is exponentially proportional to this measure. Error could also be encountered when estimating the polynomial channel matrix and this will also affect the error rate performance of the system. Future work could be undertaken to fully investigate the affect of all errors, such as channel matrix estimation error and the error obtained from truncating the polynomial matrices, on the error rate performance of the system.

8.5 Potential Applications of the PSVD

One possible application of the PSVD is in MIMO communication systems, where it can be applied to a previously estimated channel matrix to split it into a set of independent subchannels. Again, as with the application of the PQRD, the received data that has passed through the convolutive channel will be distorted due to both the effects CCI and ISI and both of these must be removed to estimate the transmitted data. By calculating the SVD of the polynomial channel matrix, provided it is of full column rank, the MIMO channel equalisation problem can be transformed into a set of single channel equalisation problems and this process

removes the CCI. The ISI present in each of the subchannels is then eliminated by applying an equaliser. For the results presented in this chapter the equalisation step has been performed using an MLSE, which operates using the Viterbi algorithm [62,66,67]. This scheme will not suffer from reduced sensitivity to frequency offset errors and has no problems with peak-to-average-power ratio, which are both problems observed when using the alternative OFDM approach.

Note that the application of the PSVD discussed here, is similar to the application of the PQRD previously discussed in 8.4.1. However, with the PQRD it is only necessary to filter either the received signals or alternatively if preferred the transmitted signals. When performing MIMO channel equalisation using the PSVD, it is necessary to filter both the received and the transmitted signals. The process of achieving broadband MIMO channel equalisation using the PSVD is now discussed.

8.5.1 MIMO Channel Exploitation

The PSVD of a polynomial channel matrix $\underline{\mathbf{C}}(z) \in \underline{\mathbb{C}}^{p \times q}$, can be calculated using either the SBR2 or the PSVD by PQRD algorithm, which have both been discussed in Chapter 7, to obtain the paraunitary matrices $\underline{\mathbf{U}}(z) \in \underline{\mathbb{C}}^{p \times p}$ and $\underline{\mathbf{V}}(z) \in \underline{\mathbb{C}}^{q \times q}$ such that

$$\underline{\mathbf{C}}(z) = \underline{\mathbf{U}}(z)\underline{\mathbf{D}}(z)\widetilde{\underline{\mathbf{V}}}(z) \tag{8.15}$$

where $\underline{\mathbf{D}}(z) \in \underline{\mathbb{C}}^{p \times q}$ is an approximately diagonal polynomial matrix.

For this application, the signals $\underline{\mathbf{s}}(z)$ demonstrated in the convolutive mixing model in equation (8.6) do not represent the source signals, but the transmitted signals, which must first be filtered at the transmitter to enable the MIMO channel equalisation problem to be transformed into a set of SISO equalisation problems. Suppose the source signals, which are generally drawn from a finite constellation such as BPSK or QPSK, are denoted by $\underline{\mathbf{s}}'(z) \in \underline{\mathbb{C}}^{q \times 1}$. Before transmitting these signals through the convolutive channel, they are passed through a transmit filter bank, where they are multiplied by the paraunitary polynomial matrix $\underline{\mathbf{V}}(z)$ obtained from calculating the PSVD of the polynomial channel matrix $\underline{\mathbf{C}}(z)$

according to equation (8.15). The filtered source signals, which are to be transmitted through the convolutive channel can therefore be expressed as

$$\underline{\mathbf{s}}(z) = \underline{\mathbf{V}}(z)\underline{\mathbf{s}}'(z). \tag{8.16}$$

As the polynomial matrix $\underline{\mathbf{V}}(z)$ is paraunitary, it will act as a multichannel all-pass filter and will therefore preserve the combined power of the signals at every frequency [16]. The filtered source signals $\underline{\mathbf{s}}(z)$ from equation (8.16) are then transmitted through the convolutive channel to obtain the received signals

$$\underline{\mathbf{x}}(z) = \underline{\mathbf{C}}(z)\underline{\mathbf{s}}(z) + \underline{\mathbf{n}}(z), \tag{8.17}$$

where $\underline{\mathbf{n}}(z) \in \underline{\mathbb{C}}^{p\times 1}$ denotes a multivariate Gaussian noise process with covariance $\sigma^2\mathbf{I}_p$ and is representative of thermal noise at the receiver. The received signal $\underline{\mathbf{x}}(z)$ are then filtered by the paraunitary polynomial matrix $\underline{\widetilde{\mathbf{U}}}(z)$, obtained from calculating the PSVD of the channel matrix $\underline{\mathbf{C}}(z)$ according to equation (8.15), to obtain the filtered received signals

$$\begin{aligned} \underline{\mathbf{x}}'(z) &= \underline{\widetilde{\mathbf{U}}}(z)\underline{\mathbf{C}}(z)\underline{\mathbf{V}}(z)\underline{\mathbf{s}}'(z) + \underline{\mathbf{n}}'(z) && (8.18)\\ &= \underline{\mathbf{D}}(z)\underline{\mathbf{s}}'(z) + \underline{\mathbf{n}}'(z) && (8.19) \end{aligned}$$

where $\underline{\mathbf{x}}'(z) = \underline{\widetilde{\mathbf{U}}}(z)\underline{\mathbf{x}}(z)$ and $\underline{\mathbf{n}}'(z) = \underline{\widetilde{\mathbf{U}}}(z)\underline{\mathbf{n}}(z)$. Furthermore, as the polynomial matrix $\underline{\widetilde{\mathbf{U}}}(z)$ is paraunitary, $\underline{\mathbf{n}}'(z)$ is also a Gaussian noise process with identical spectral properties. Equations (8.18) and (8.19) have demonstrated that passing the source signals through the paraunitary polynomial matrix $\underline{\mathbf{V}}(z)$, then through the convolutive channel $\underline{\mathbf{C}}(z)$ and finally through the paraunitary polynomial matrix $\underline{\widetilde{\mathbf{U}}}(z)$ is equivalent passing the signals through the approximately diagonal polynomial matrix $\underline{\mathbf{D}}(z)$. In particular, due to the diagonal structure of $\underline{\mathbf{D}}(z)$, the i^{th} received signal, where $i = 1, \ldots, q$, can be written as

$$\underline{x}_i'(z) = \underline{d}_{ii}(z)\underline{s}_i'(z) + \underline{n}_i'(z), \tag{8.20}$$

which is a single channel equalisation problem and can be solved using an MLSE to obtain an estimate of the i^{th} source signal $\underline{s}_i'(t)$. Note that when using the PQRD to simplify the MIMO channel equalisation problem, due to the structure of the upper triangular polynomial matrix $\underline{\mathbf{R}}(z)$, the source signals must be estimated in a particular order and the SISO equation relating to a particular source, will require knowledge of all sources that have been previously estimated. However, when using the PSVD this is no longer the case. The estimated sources can be determined in any order and all of the SISO problems are independent of the other source signals. A block diagram of the proposed baseband communication system using the PSVD can be seen in Figure 8.3.

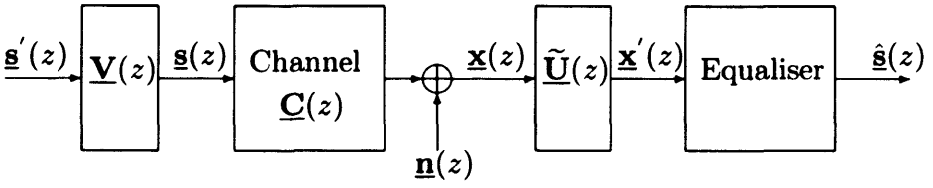

Figure 8.3: Block diagram for a basic baseband communication system using the PSVD (schematic of the multi-channel exploitation scheme).

If the channel matrix $\underline{\mathbf{C}}(z) \in \underline{\mathbb{C}}^{p \times q}$ is rank deficient, then it will still be possible to successfully transmit and obtain an estimate over a number of the subchannels. In this situation, the PSVD will generate a diagonal polynomial matrix where a number of the diagonal elements are equal to zero. Consequently, the signals transmitted over these channels cannot be estimated using this method. However, unlike the method of channel equalisation using the PQRD discussed in the previous section, it is still possible to transmit and receive over all of the other subchannels, which are associated with the non-zero diagonal elements.

A considerable amount of research has already been undertaken in this area [11–14]. However, all research so far has focused on calculating the PSVD using the PEVD routine SBR2. Furthermore, the results are often demonstrated for exponentially decaying profile channels, which will generally result is a diagonal polynomial matrix $\underline{\mathbf{D}}(z)$ of low order, but these channel matrices are not representative of a typical wireless channel. The following

numerical examples to illustrate this application, will calculate the PSVD using both this existing SBR2 method and also the new PQRD technique introduced in Chapter 7. In this chapter, the PSVD by PQRD algorithm has demonstrated better performance when applied to a single polynomial matrix. The main advantage of this algorithm is that it is possible to specify how small the coefficients associated with the off-diagonal polynomial elements must be driven, which is something that it is not possible to do when using the SBR2 method. As a result, the PSVD by PQRD algorithm typically requires a smaller amount of computational time to reach the same level of decomposition as the PSVD by PEVD method. Furthermore, the orders of the polynomial matrices and the relative error of the decomposition are generally less when using this method. The channel matrices for these examples are also chosen to have constant power profile, as this is representative of a typical wireless channel. The following two examples compare the two decomposition techniques, illustrating the advantages of the PQRD over the PEVD method. For both examples MIMO channel equalisation, as discussed here, is then carried out using the results obtained from the PSVD by PQRD algorithm.

8.5.2 Numerical Examples

The two examples used in Section 8.4.3, which have previously been used to demonstrate the potential application of the PQRD, are now used to demonstrate the possible application of the PSVD to MIMO communication systems. However, the two polynomial channel matrices are first used as further examples of the PSVD by PQRD algorithm discussed in Chapter 7. These examples serve to illustrate the improved performance of this algorithm over the PSVD by PEVD method, which was previously used for this application [11–14, 72, 73].

Note that it is hard to compare the two algorithms based on their computational complexity due to the growing orders of the polynomial matrices, which can not be determined in advance. Therefore, the best measure available is the computational time taken to form the decomposition using each algorithm. Furthermore, it is also difficult to achieve the same level of decomposition using the two different techniques, as each algorithm will only formulate an approximation and it is impossible to specify in advance how small the off-diagonal elements should be driven when using the SBR2 method. This is due to the PSVD by PEVD method

not directly calculating the PSVD of $\underline{\mathbf{A}}(z)$, but instead formulating the PSVD by calculating the PEVD of the two para-Hermitian polynomial matrices $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$ and $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$ to obtain the left and right hand polynomial singular vectors respectively. Instead, a process of trial and error must be undertaken. This will then reflect in the relative error of the decomposition and also the final value of g, which defines the magnitude of the dominant coefficient of the approximately diagonal polynomial matrix $\underline{\mathbf{D}}(z)$ obtained from the PSVD. The size of the relative error will influence the transmission symbol BER for the system. This is not a problem when using the PSVD by PQRD method, where the stopping condition directly specifies the accuracy of the polynomial matrix decomposition. This point was discussed in more detail in Chapter 7.

For each example, the elements of the diagonal polynomial matrix, obtained using each method, are aligned to ensure that they are over the same series of lags. This was previously done for the worked example in Chapter 7. Note that to realign the diagonal elements a series of delay matrices can be either applied from the left or right hand side of the polynomial matrix. This will be determined by which rows or columns of the the paraunitary transformation consists of elements that must also be realigned. This step will therefore not only enable the possibility of reducing the order of the diagonal matrix, but also the orders of one or possibly both of the paraunitary transformation matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$.

Example 1

Firstly, the two methods for calculating the PSVD are applied to the polynomial channel matrix $\underline{\mathbf{C}}_1(z) \in \mathbb{C}^{4\times3\times4}$. The details of how this polynomial matrix was generated can be found in Section 8.4.3, where it was previously used to demonstrate the potential application of the PQRD. The PQRD-BC algorithm was used to calculated the PSVD as it required the least number of iterations when compared to the PSVD using either the PQRD-BS or the PQRD-SBR variations of the algorithm. Consequently, it required the least amount of computational time. Note that this method was also the best algorithm to use when calculating the PQRD of the same channel matrix. The results from applying the PSVD by PEVD and the PSVD by PQRD-BC algorithms, with the truncation and stopping parameters

set as $\epsilon = 10^{-2}$ and $\mu = 10^{-6}$, can be seen in Table 8.5. For these results a final alignment step, as discussed in Chapter 7, was applied to the polynomial matrices, if required, to further reduce the orders of the polynomial matrices.

	PSVD By	
	SBR2	**PQRD**
Number of Iterations	109	456
g	5.65×10^{-2}	9.82×10^{-3}
Order of $\underline{\mathbf{D}}(z)$	56	57
Order of $\underline{\mathbf{U}}(z)$	47	61
Order of $\underline{\mathbf{V}}(z)$	29	52
E^{rel}	0.2180	0.0862
Computational Time (Seconds)	0.25	0.60

Table 8.5: Results obtained from applying the two methods for calculating the PSVD to the polynomial matrix $\underline{\mathbf{C}}_1(z)$.

The results presented in this table demonstrate that the PSVD by PQRD algorithm took more iterations and time to converge than the PEVD method. Furthermore, the order of the resulting polynomial matrices $\underline{\mathbf{D}}(z)$, $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ obtained by the decomposition were also larger. However, the PSVD by PQRD algorithm obtained a far more accurate decomposition due to the off-diagonal coefficients being driven significantly smaller according to the stopping criterion ϵ. The magnitude of the largest coefficient associated with an off-diagonal polynomial element of $\underline{\mathbf{D}}(z)$ was found to be $g = 9.82 \times 10^{-3}$. The same level of decomposition was not obtained using the PSVD by PEVD algorithm, where $g = 5.65 \times 10^{-2}$. Furthermore, an upper bound on this value could not be determined in advance using this method and so the same problems observed in the numerical example in Chapter 7 were again present.

To obtain a similar level of decomposition as obtained by the PSVD by PQRD algorithm required the truncation and stopping parameters to be set as $\mu = 10^{-7}$ and $\epsilon = 10^{-3}$. The algorithm now required 0.91 seconds to converge to a point where $g = 6.72 \times 10^{-3}$, which did not take into account the time taken to find the appropriate values of μ and ϵ to obtain

this level of decomposition. Furthermore, the order of the approximately diagonal matrix is now 67. Comparing these results with those obtained using the PSVD by PQRD algorithm as presented in Table 8.5 confirmed that the PSVD by PQRD algorithm is faster to obtain the decomposition and the orders of the polynomial matrices obtained for this decomposition are shorter. For this reason PSVD by PQRD algorithm has been used to perform the MIMO channel equalisation.

Subsequently, a set of three BPSK source signals each of length 1000 were generated and then filtered by the paraunitary polynomial matrix $\underline{\mathbf{V}}(z)$ obtained from calculating the PSVD of the channel matrix $\underline{\mathbf{C}}_1(z)$. The filtered source signals are then convolutively mixed using the channel matrix $\underline{\mathbf{C}}_1(z)$ according to equation (2.28), where N defines the order of the polynomial channel matrix and for this example is equal to four. Gaussian noise representative of thermal noise, with spatial covariance $\sigma^2\mathbf{I}_4$, was then added to each of the receive sensors to give a desired RSNR, which is again calculated according to equation (8.14). The received signals are then filtered by the paraunitary polynomial matrix $\underline{\tilde{\mathbf{U}}}(z)$, which was also obtained from calculating the PSVD of the channel matrix $\underline{\mathbf{C}}_1(z)$, and the process of equalisation of each filtered received signal was performed to obtain estimates of each of the three source signals. The average BER for each of the estimated source signals was calculated, where the variance of the additive noise was chosen to give varying levels of RSNR, ranging from -5 to 20dB. This was carried out for 100 independent realisations, using the same channel matrix $\underline{\mathbf{C}}_1(z)$, but generating new source signals and noise terms for each realisation. The average of these results can be seen in Table 8.6, where for RSNR levels 10 to 20 dB the technique offers excellent error rate performance for the first source. The error performance of the remaining two sources is not so good, especially for the third source.

Example 2

Both techniques for calculating the PSVD were applied to the polynomial matrix $\underline{\mathbf{C}}_2(z) \in \mathbb{C}^{5\times5\times4}$. Again, the PQRD-BC algorithm was used for the PSVD by PQRD algorithm as it provided the best performance for this example. The PQRD-BC algorithm required less time to converge, the relative error was less than that obtained using the other algorithms and

SNR (dB)	Average BER							
	$\underline{\mathbf{C}}_1(z)$			$\underline{\mathbf{C}}_2(z)$				
	Source 1	Source 2	Source 3	Source 1	Source 2	Source 3	Source 4	Source 5
−5	0.1979	0.2501	0.3804	0.1867	0.2490	0.3142	0.3735	0.4391
0	0.0638	0.1201	0.3003	0.0578	0.1167	0.1943	0.2871	0.3885
5	0.0032	0.0175	0.1801	0.0026	0.0166	0.0647	0.1673	0.3100
10	0	0.0001	0.0512	0	0.0001	0.0035	0.0422	0.1932
15	0	0	0.0022	0	0	0	0.0011	0.0684
20	0	0	0	0	0	0	0	0.0062

Table 8.6: Average BERs for a set of estimated BPSK sources for the MIMO channel equalisation problem calculated using the PSVD by PQRD-BC algorithm to split the problem into a set of SISO problems, which can then equalised using a MLSE based on the Viterbi algorithm. The results are demonstrated for the two channel matrices $\underline{\mathbf{C}}_1(z) \in \mathbb{C}^{4\times3\times4}$ and $\underline{\mathbf{C}}_2(z) \in \mathbb{C}^{5\times5\times4}$, and for different levels of RSNR ranging from −5dB to 20dB.

the orders of the polynomial matrices obtained by the algorithm were also shorter. A final alignment step was applied to the results obtained using both PSVD techniques, reducing the orders of the polynomial matrices at no additional cost to the relative error. The results can be seen in Table 8.7.

These results demonstrate that the PQRD technique obtains a far more accurate decomposition, with $E^{rel} = 0.1261$. The same measure found using the PEVD method was found to be 0.4146. However, the PQRD method did require considerably more time to converge, but once again the magnitude of the largest coefficient associated with an off-diagonal polynomial element of $\underline{\mathbf{D}}(z)$ was found to be $g = 9.93 \times 10^{-3}$, much smaller that that obtained using the PEVD method.

To obtain a similar level of decomposition to that obtained by the PSVD by PQRD algorithm, required a considerable amount of trial and error to find an appropriate choice of the parameters μ and ϵ to obtain a similar value of g. If $\epsilon = 5 \times 10^{-4}$ and $\mu = 10^{-7}$, then the magnitude of the dominant off-diagonal coefficient of $\underline{\mathbf{D}}(z)$ was now found to be $g = 9.04 \times 10^{-3}$. However, this required 1239 iterations, taking 5.93 seconds to converge and

	PSVD By	
	SBR2	**PQRD**
Number of Iterations	168	869
g	6.72×10^{-2}	9.93×10^{-3}
Order of $\underline{\mathbf{D}}(z)$	47	78
Order of $\underline{\mathbf{U}}(z)$	38	77
Order of $\underline{\mathbf{V}}(z)$	32	54
E^{rel}	0.4146	0.1261
Computational Time (Seconds)	0.50	1.37

Table 8.7: Results obtained from applying the two methods for calculating the PSVD to the polynomial matrix $\underline{\mathbf{C}}_2(z)$.

the order of the approximately diagonal polynomial matrix $\underline{\mathbf{D}}(z)$ was 132. Therefore, the PSVD by PQRD clearly outperforms the PSVD by PEVD algorithm. For this reason this algorithm was used to perform the MIMO channel equalisation.

The process of equalisation is exactly the same for the previous example and so will not be discussed again. For this example, there are five sources signals, which were chosen to be BPSK sequences of length 1000. The average error rate results were averaged over 100 independent realisations, using the same channel matrix $\underline{\mathbf{C}}_2(z)$, but generating new source signals and noise terms for each realisation. The average of these results can be seen in Table 8.6, where for RSNR levels 10 to 20 dB the technique offers excellent error rate performance for the first source. As with the previous example, the error rate performance will degrade moving downwards through the sources. In particular, the error performance of the fifth source was poor for all levels of RSNR.

8.5.3 Conclusions

The numerical examples have demonstrated the ability of the PSVD by PQRD algorithm as a preprocessing step in MIMO communication systems to transform a problem of MIMO channel equalisation into a series of SISO channel problems, which can be solved using an equalisation scheme such as MLSE. The polynomial channel matrices used for these results

were chosen to have constant power profile to represent a typical wireless channel and the scheme has demonstrated good average error rate performance for these channels.

The results have also illustrated the improved performance of using the PSVD by PQRD algorithm over the existing technique of using the SBR2 algorithm to calculate the PSVD of a polynomial matrix. Again as with the simulated results to demonstrate the application of the PQRD, only a very simple baseband communication system has been demonstrated and so the average error rate could potentially be improved by using an interleaver, a different method of equalisation and possibly also by implementing error correction coding [62, 71].

8.5.4 Paraunitary Filter Bank Design and Subband Coding

The PSVD can also be used with subband coding, which is extensively described in [9, 10]. The idea of subband coding is to split a signal into a number of different subbands, which can then each be individually decimated, with the allocation of bits per subband being determined by the energy content of each subband signal [16, 57, 58]. It is used extensively for the data compression of audio signals - for example to generate MPEG audio files. Consequently, this will conserve signal bandwidth, by eradicating any information concerning frequencies that won't noticeably change the reconstructed signals. In [9, 10] the SBR2 algorithm has been used to calculate the PSVD. In [10] a slightly modified version of the SBR2 algorithm is used, referred to as the SBR2 coder, to obtain the paraunitary matrix filter bank. The PSVD calculated the optimal FIR paraunitary filterbank.

8.6 Potential Applications of the PEVD

The application of the PEVD to strong decorrelation has been discussed extensively in Section 3.6.1. This was the primary application of the decomposition, when it was developed to form the first step of a two-step BSS algorithm suitable for convolved signals. Since then other applications of the decomposition have been realised, including the potential application to MIMO communications discussed in 8.5.1, where the PEVD can be used to formulate the PSVD.

Chapter 9

Conclusions and Future Work

The main limitation of the SBR2 algorithm was the unnecessarily large orders of the resulting polynomial matrices generated by the algorithm. The first contribution discussed in this thesis, was the development of an energy based truncation method, which can allow the order of these polynomial matrices to be vastly reduced whilst, if used appropriately, still maintaining an accurate polynomial matrix decomposition. This then enabled the computational load of the SBR2 algorithm to be reduced, which consequently meant that the algorithm was typically faster to run. This is illustrated by the examples detailed in the fourth chapter, demonstrating the orders of the resulting polynomial matrices and the computational time taken to run the SBR2 algorithm can be vastly reduced when this truncation method is included. This result is therefore useful for the potential applications of the algorithm, where the resulting orders of the matrices is critical. In particular, this can be used to great advantage for the application of the decomposition to MIMO communication problems where the computational complexity required to solve the problem is directly proportional to the order of the diagonal polynomial matrix generated by the SBR2 algorithm.

Subsequently, three algorithms for calculating the QR decomposition of a polynomial matrix have been introduced, all of which are guaranteed to transform a polynomial matrix into an approximately upper triangular polynomial matrix by means of polynomial paraunitary matrices. Results have demonstrated the most efficient of these algorithms to generally be the PQRD by Columns (PQRD-BC) algorithm, although for some examples the PQRD by Steps (PQRD-BS) algorithm does outperform this method. All three algorithms introduced

for calculating this decomposition have been proven to converge. The potential application of this decomposition is to MIMO communication problems, where it is often required to reconstruct data sequences that have been distorted due to the effects of co-channel interference and multipath propagation, leading to intersymbol interference. If the polynomial channel matrix for the system is known, its QR decomposition can be calculated using one of these algorithms and this can then be used to transform the linear system of polynomial equations into triangular form, which can then be solved using back substitution and a standard equalisation technique suitable for single channel problems. This process is extensively discussed in the penultimate chapter, where some simulated average bit error rate results are presented to support this potential application. The energy based truncation method can again be used to great advantage to this application of the decomposition algorithm. Note that the possible applications of this decomposition to broadband signal processing will be as diverse as the applications of the scalar matrix QRD to narrowband signal processing.

An algorithm for calculating the SVD of a polynomial matrix has also been presented. The algorithm operates by iteratively applying the most efficient of the PQRD algorithms and is therefore referred to as the PSVD by PQRD algorithm. A proof of convergence for the algorithm has been presented in this thesis. This algorithm has been compared to an already existing technique for calculating the SVD of a polynomial matrix, which operates by applying the SBR2 algorithm (referred to as the PSVD by PEVD algorithm), and numerical results have demonstrated the PSVD by PQRD algorithm offers better performance. The main advantage of the PSVD by PQRD algorithm, is that it is computationally considerably faster to obtain approximately the same level of decomposition. Secondly, the PSVD by PQRD algorithm also allows the user control over how small the off-diagonal elements of the matrix must be driven before convergence has been reached, which is something that cannot be achieved using the SBR2 approach without a considerable amount of trial and error. The resulting orders of the polynomial matrices obtained using the PSVD by PQRD method are typically shorter than those obtained with the PSVD by PEVD approach, with less relative error. This final point can be an advantage for the potential application of the decomposition to MIMO communications and has been discussed in detail in chapter eight. Note that the

polynomial matrix truncation methods can also be used within these new decompositions algorithms.

The main contributions of the thesis are the algorithms for calculating the QR and singular value decomposition of a polynomial matrix. They have all been proven to converge and are also numerically robust. The thesis has outlined the potential applications of these decompositions and provided some average error rate result to support the applications. The energy based truncation method, which can be used to reduce the computational requirements within any of the polynomial matrix decomposition algorithms is also a contribution of the thesis.

9.1 Suggestions for Further Work

This section has been subdivided into the potential areas of research for the three decomposition discussed in this thesis.

The PEVD

The classical Jacobi algorithm for calculating the EVD of a scalar matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$ involves $O(n^2)$ operations to search for the dominant element in the matrix at each iteration and other aspects of the iteration require $O(n)$ operations [6]. Alternatively, a cyclic by rows Jacobi algorithm can be used to reduce the number of computations, by visiting the off-diagonal elements in the matrix using an ordering, implementing multiple sweeps if required for convergence. This algorithm is considerably faster as it does not require a search routine to locate the dominant element at each iteration. It would be interesting to see if a similar approach could be used with the SBR2 algorithm and how this would affect convergence of the algorithm. Furthermore, it would be interesting to see if parallel computations could be used within the SBR2 algorithm, by implementing non-conflicting Givens rotations in parallel. For example, the rotations required to zero coefficients in the polynomial elements $(1,2)$ and $(3,4)$ are non-conflicting and can therefore be carried out in parallel. However, with polynomial matrices this will require multiple applications of delay matrices, which may

cause problems for the same reason that Householder refections are not a suitable type of transformation for polynomial matrices[1].

The PQRD

For scalar matrices, the LU decomposition is generally preferable to the QR decomposition when solving a set of linear equations, as it requires approximately half the number of operations to calculate [38]. In fact for the scalar matrix case, there are a few exceptions where the QRD is the best method to use. Could an algorithm for calculating the LU decomposition of a polynomial matrix be developed using similar methods and techniques to those discussed in this thesis? Furthermore, if this decomposition is possible, would it provide a less computationally expensive algorithm for polynomial matrices than the PQRD? It is difficult to determine this in advance due to the orders of the polynomial matrices growing at each iteration due to the application of elementary delay matrices, which are a necessity for convergence of the polynomial matrix decompositions.

The QR algorithm of a scalar matrix is often used to calculate the eigenvalues of a matrix, where the EVD can not be used as the matrix is not Hermitian. Parallel Givens rotations could be applied, which may affect both the order of the polynomial matrices and the computational time. Furthermore, there exist variations formulating the Givens method for scalar matrix QRD, such as the Fast Givens QRD and techniques using column pivoting. The focus of future research could investigate these possibilities, to see if they are feasible.

The application of the PQRD to MIMO communication systems, discussed in chapter eight, was presented for a very simple baseband communication systems, yet the technique still demonstrated good average error rate performance. The use of interleavers and error correction codes could be used to potentially improve these results. Furthermore, different methods of equalisation could also be applied to the problem.

[1]Householder reflections suitable for polynomial matrix decompositions are discussed in Appendix A.

The PSVD

With scalar matrix decompositions, the SVD can be used to calculate the pseudo-inverse of a matrix. Could an algorithm be developed for calculating the pseudo-inverse of a polynomial matrix using the PSVD? This would be useful for problems where the polynomial matrix is rank deficient and so the PQRD can not be used to solve a set of polynomial linear equations as discussed in chapter eight. Finally, as with the PQRD future work, the numerical results discussed in chapter eight could potentially be improved by using a different method of equalisation, interleavers or possibly error correction coding could be used to potentially improve these results.

Bibliography

[1] A. Peled and B. Liu. *Digital Signal Processing.* John Wiley and Sons, 1976.

[2] A.J. Paulraj, R. Nabar and D. Gore. *Introduction to Space-Time Wireless Communications.* Cambridge University Press, 2003.

[3] A. Hyvärinen, J. Karhunen and E. Oja. *Independent Component Analysis.* John Wiley and Sons, 2001.

[4] S. Haykin. *Unsupervised Adaptive Filtering, Volume 1: Blind Source Separation.* John Wiley and Sons, 2000.

[5] P. Comon. Independent Component Analysis, a New Concept? *Signal Processing, Elsevier*, 36(3):287–314, 1992.

[6] G.H. Golub and C.F. Van Loan. *Matrix Computations (Third Edition).* The John Hopkins University Press, 1996.

[7] 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*, 55(6):2158–2169, 2007.

[8] P.D. Baxter. *Blind Signal Separation of Convolutive Mixtures.* PhD thesis, Department of Electrical and Electronic Engineering, Imperial College, 2005.

[9] S. Redif. *Polynomial Matrix Decompositions and Paraunitary Filter Banks.* PhD thesis, School of Electronics and Computer Science, University of Southampton, 2006.

[10] S. Redif and T. Cooper. Paraunitary filterbank design via a polynomial singular value decomposition. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 613–616, 2005.

[11] C.H. Ta and S. Weiss. A Design of Precoding and Equalisation for Broadband MIMO Transmission. In 15^{th} *International Conference on Digital Signal Processing, Cardiff*, 2007.

[12] C.H. Ta and S. Weiss. A Design of Precoding and Equalisation for Broadband MIMO Systems. In 41^{st} *Asilomar Conference on Signals, Systems and Computers, Asilomar, USA*, 2007.

[13] M. Davies, S. Lambotharan and J. G. McWhirter. Polynomial Matrix SVD for MIMO Broadband Beamforming. In *IMA Conference on Mathematics in Signal Processing, Cirencester*, 2006.

[14] M. Davies, S. Lambotharan and J. G. McWhirter. Broadband MIMO Beamforming using Spatial-Temporal Filters and Polynomial Matrix Decomposition. In 15^{th} *International Conference on Digital Signal Processing, Cardiff*, 2007.

[15] M. Davies, S. Lambotharan, J. Chambers and J. G. McWhirter. Broadband MIMO Beamforming For Frequency Selective Channels Using the Sequential Best Rotation Algorithm. In 67^{th} *Vehicular Technology Conference, Singapore*, May 2008.

[16] P.P. Vaidyanathan. *Multirate Systems and Filter Banks.* Prentice Hall, 1993.

[17] S. Weiss, S. Redif, T. Cooper, C. Liu, P.D. Baxter and J.G. McWhriter. Paraunitary oversampled filterbank design for channel coding. *EURASIP Journal of Applied Signal Processing*, 2006:1–10, 2006.

[18] T. Kailath. *Linear Systems.* Prentice-Hall International Inc., 1980.

[19] X. Gao, T.Q. Nguyen and G. Strang. On Factorization of M-Channel Paraunitary Filterbanks. *IEEE Transactions on Signal Processing*, 49(7):1433–1446, 2001.

[20] R.H. Lambert. *Multichannel Blind Deconvolution: FIR Matrix Algebra and Separation of Multipath Mixtures.* PhD thesis, Department of Electrical Engineering, University of Southern California, 1996.

[21] J. G. McWhirter and P. D. Baxter. A Novel Technique for Broadband SVD. In 12^{th} *Annual Workshop on Adaptive Sensor Array Processing*, 2004.

[22] L. De Lathauwer, B. De Moor and J. Vandewalle. An Algebraic Approach to Blind MIMO Identification. In *Proc. of the* 2^{nd} *International Workshop on Independent Component Analysis and Blind Source Separation (ICA 2000)*, pages 211–214, 2000.

[23] L. De Lathauwer, B. De Moor and J. Vandewalle. Fetal Electrocardiogram Extraction by Blind Source Subspace Separation. *Proc. of the IEEE Transactions on Biomedical Engineering, Special Topic Section on Advances in Statistical Signal Processing for Biomedicine*, (9):567–572, 2000.

[24] S. Makeig, A. Bell, T.P. Jung and T.J. Sejnowski. Independent Component Analysis of Electroencephalographic Data. *Advances in Neural Information Processing Systems*, 1995.

[25] S. Haykin. *Adaptive Filter Theory (Fourth Edition).* Prentice Hall, 2002.

[26] S. Choi, A. Cichocki, H.M Park and S.Y. Lee. Blind Source Separation and Independent Component Analysis: A Review. *Neural Information Processing - Letters and Reviews*, 6(1):1–57, 2005.

[27] A. Hyvärinen and E. Oja. Independent Component Analysis: Algorithms and Applications. *Neural Networks*, 13(4):411–430, 2000.

[28] J.F. Cardoso. Blind Signal Separation: Statistical Principles. *Proceedings of the IEEE*, 86(10):2009–2025, 1998.

[29] J.F. Cardoso and A. Souloumiac. Blind Beamforming for Non Gaussian Signals. *Radar and Signal Processing, IEE Proceedings F*, 140(6):362–370, 1993.

[30] A. Belouchrani, J. Abed-Meraimm, J. Cardoso and EE. Moulines. A Blind Source Separation Tehcnique using Second-order Statistics. *IEEE Transactions on Signal Processing*, 45(2):434–444, 1997.

[31] L. De Lathauwer, B. De Moor and J. Vandewalle. Blind Source Separation by Simultaneous Third-order Tensor Diagonalisation. In *Proc. of the* 8^{th} *European Signal Processing Conference (EUSIPCO '96)*, pages 91–96, 1996.

[32] I. J. Clarke. Direct Exploitation of Non Gaussianity as a Discriminant. *Proc. EUSIPCOIX*, pages 2057–2060, 1998.

[33] A. Hyvärinen and E. Oja. A Fast Fixed-Point Algorithm for Independent Component Analysis. *Neural Computation*, 9:1483–1492, 1997.

[34] K.J. Pope and R.E. Bogner. Blind Signal Separation I. Linear, Instantaneous Combinations. *Digital Signal Processing*, 6:5–16, 1996.

[35] G.W. Stewart. *Computer Science and Applied Mathematics.* Academic Press, 1973.

[36] G. Strang. *Linear Algebra and its Applications.* Academic Press Inc., 1976.

[37] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, 1965.

[38] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery. *Numerical Receipes in C, The Art of Scientific Computing (Second Edition).* Cambridge University Press, 2002.

[39] M. H. Hayes. *Statistical Digital Signal Processing and Modeling.* John Wiley and Sons, 1996.

[40] G.W. Stewart. On the Early History of the Singular Value Decomposition. *SIAM Review*, 35(4):551–566, 1992.

[41] Edited by M. Moonen and B. De Moor. *SVD and Signal Processing, III.* Elsevier, 1995.

[42] E.R. Dougherty and C.R. Giardina. *Matrix Structured Image Processing.* Prentice Hall, 1987.

[43] T. Lee, A.J. Bell and R.H. Lambert. Blind Separation of of Delayed and Convolved Sources. *Advances in Neural Information Processing Systems*, pages 758–764, 1997.

[44] K. Torkkola. Blind Separation of Convolved Sources Based on Informationmaximization. In *Proc. of the IEEE Neural Networks for Signal Processing*, pages 423–432, 1996.

[45] M. S. Pedersen, J. Larsen, U. Kjems and L. C. Para. A Survey of Convolutive Blind Source Separation Methods. *Springer Handbook on Speech Processing and Speech Communication.*

[46] K.J. Pope and R.E. Bogner. Blind Signal Separation II. Linear, Convolutive Combinations. *Digital Signal Processing*, 6:17–28, 1996.

[47] P. Smaragdis. Blind Separation of Convolved Mixtures in the Frequency Domain. In *International Workshop on Independence and Artificial Neural Networks, Spain*, 1998.

[48] R. Klemm. *Space-time Adaptive Processing: Principles and Applications*, volume 9 of *IEE Radar, Sonar, Navigation and Avionics.* 1998.

[49] A.J. Paulraj and C.B. Papadias. Space-Time Processing for Wireless Communications. *IEEE Signal Processing Magazine*, 14:49–83, 1997.

[50] P.A. Regalia and D.-Y Huang. Attainable Error Bounds in Multirate Adaptive Lossless FIR Filters. In *IEEE Conference Acoustic, Speech and Signal Processing*, pages 1460–1463, 1995.

[51] H.J.S. Smith. On Systems of Linear Indeterminate Equations and Congruences. *Philosophical Transactions of the Royal Society London*, 151:293–326, 1861.

[52] R. H. Lambert, C. L. Nikias. Polynomial Matrix Whitening and Application to the Multichannel Blind Deconvolution Problem. *Military Communications Conference*, 3:988–992, 1995.

[53] R. H. Lambert, M. Joho and H. Mathis. Polynomial Singular Values for Number of Wideband Sources Estimation and Principal Component Analysis. In *Proc. Int. Conf. Independent Component Analysis*, pages 379–383, 2001.

[54] P. D. Baxter and J. G. McWhirter. Blind signal separation of convolutive mixtures. In *Proc. 37^{th} Asilomar Conference on Signals, Systems and Computers*, 2003.

[55] C.H. Ta and S. Weiss. Design of Precoding and Equalisation for Broadband MIMO Transmission. In *Second IEE/EURASIP Conference on DSP enabled Radio*, 2005.

[56] J.A.Cadzow. *Foundations of Digital Signal Processing and Data Analysis.* Collier Macmillan, 1987.

[57] P.P. Vaidyanathan. Theory of Optimal Orthonormal Subband Coders. *IEEE Transactions on Signal Processing*, 46(6):1528–1543, 1998.

[58] P.P. Vaidyanathan and A. Kirac. Results on Optimal Biorthogonal Filter Banks. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(8):932–947, 1998.

[59] J. Foster, J. G. McWhirter and J. Chambers. Limiting the Order of Polynomial Matrices Within the SBR2 Algorithm. In *IMA Conference on Mathematics in Signal Processing, Cirencester*, 2006.

[60] A.J. Paulraj, D. Gore, R.U. Nabar and H. Bölcskei. An Overview of MIMO Communications - A Key to Gigabit Wireless. *Proceedings of the IEEE*, 92(2):198–218, 2004.

[61] D. Gesbert, M. Shafi, D.S. Shiu, P. Smith and A. Naguib. From Theory to Practice: An Overview of MIMO Space-Time Coded Wireless Systems. *IEEE Journal on Selected Areas in Communications. Special Issue on MIMO Systems, part 1*, 21:281–302, 2003.

[62] J.G. Proakis. *Digital Communications (Fourth Edition).* McGraw-Hill Book Co., 2001.

[63] L. Tong, G. Xu and T. Kailath. A New Approach to Blind Identification and Equalisation of Multipath Channels. In *Proc. 25^{th} Asilomar Conference on Signals, Systems and Computers*, pages 856–860, 1991.

[64] C. Berrou and A. Glavieux. Near Opitimum Error Correcting Coding and Decoding: Turbo Codes. *IEEE Transactions on Communications*, pages 1261–1271.

[65] A.J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[66] A.J. Viterbi. A Personal History of the Viterbi Algorithm. *Signal Processing Magazine, IEEE*, 23(4):120–142, 2006.

[67] G.D. Forney Jr. The Viterbi Algorithm. *Proc. IEEE*, 61:268–278, 1973.

[68] R.O. Duda, P.E. Hart and D.G. Stork. *Pattern Classification (Second Edition).* John Wiley Sons Inc., 2001.

[69] C. Burge and S. Karlin. Prediction of Complete Gene Structures in Human Genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

[70] J. Foster, J. G. McWhirter and J. Chambers. A Polynomial Matrix QR Decomposition with Application to MIMO Channel Equalisation. In *Proc.* 41^{st} *Asilomar Conference on Signals, Systems and Computers*, 2007.

[71] B.P. Lathi. *Modern Digital and Analog Communication Systems (Third Edition).* Oxford University Press, 1998.

[72] C. Liu, S. Weiss, S. Redif, T. Cooper, L. Lampe and J.G. McWhriter. Channel Coding for Power Line Communication Based on Oversampled Filter Banks. In *2005 International Symposium on Power Line Communications and Its Applications*, 2005.

[73] S. Weiss, C.H. Ta and C. Liu. A Wiener Filter Approach to the Design of Filter Bank Based Single-Carrier Precoding and Equalisation. In *Power Line Communications and Its Applications*, pages 493–498, 2007.

Appendix A

Householder Transformations for Polynomial Matrices

When calculating the QRD of a scalar matrix, Householder transformations (also referred to as Householder reflections or elementary reflectors) are often used as an alternative to Givens rotations, as they allow zeros to be introduced to a matrix on a grand scale rather than the very selective procedure observed when using Givens rotations [6]. Furthermore, Householder reflections are typically computationally less expensive when calculating the QRD of a scalar matrix. Givens rotations are still useful for scalar matrix decompositions, for example they are more appropriate when calculating the decomposition of a sparse matrix and it is also easier to run parallel computations with Givens rotations. This appendix examines the concept of applying Householder reflections as an alternative to Givens rotations as a technique for achieving a polynomial matrix QRD. Firstly, the conventional Householder reflections applicable to scalar matrices are discussed.

A.1 Householder Reflections for Scalar Matrices

A.1.1 Real Householder Reflections

Suppose for a vector of scalars $\mathbf{x} \in \mathbb{R}^{p\times 1}$, where $\mathbf{x} = [x_1, \ldots, x_p]^T \neq \mathbf{0}$, we wish to zero all elements beneath the first element x_1. This could be achieved by applying a series of Givens rotations to drive each element in turn to zero, or alternatively could be accomplished by applying one Householder reflection which will drive all of the elements beneath the first

element x_1 to zero directly. A Householder reflection is a matrix of the form

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\|\mathbf{v}\|_2^2} \tag{A.1}$$

where

$$\mathbf{v} = \mathbf{x} \pm \|\mathbf{x}\|_2 \mathbf{e}_1 \tag{A.2}$$

and $\mathbf{e}_1 = [1, 0, \ldots, 0]^T \in \mathbb{C}^{p\times 1}$, i.e. the first canonical vector [6]. This matrix can be applied to the vector $\mathbf{x}$ resulting in the transformation

$$\mathbf{H}\mathbf{x} = \mp \|\mathbf{x}\|_2 \mathbf{e}_1 \tag{A.3}$$

and so all elements of the transformed vector beneath the first element x_1 are now equal to zero. Furthermore, as with Givens rotations, the transformation matrix $\mathbf{H}$ is unitary and so the transformation is norm-preserving, i.e. $\|\mathbf{H}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$.

Householder reflections can similarly be applied to a matrix $\mathbf{A} \in \mathbb{R}^{p\times q}$, whose elements are real scalars. The required Householder reflection to zero all elements beneath the diagonal of the k^{th} column takes the form of a $p\times p$ identity matrix with the exception of the $(p-k)\times(p-k)$ submatrix $\widehat{\mathbf{H}}_k$ formulated as

$$\widehat{\mathbf{H}}_k = \mathbf{I}_{(p-k)} - 2\frac{\mathbf{v}\mathbf{v}^T}{\|\mathbf{v}\|_2^2} \tag{A.4}$$

where

$$\mathbf{v} = \mathbf{a}_k \pm \|\mathbf{a}_k\|_2 \mathbf{e}_1, \tag{A.5}$$

$\mathbf{a}_k = [a_{kk}, a_{(k+1)k}, \ldots, a_{pk}]^T$ and $\mathbf{e}_1 = [1, 0, \ldots, 0]^T \in \mathbb{R}^{(p-k)\times 1}$. The entire Householder reflection matrix will be denoted as $\mathbf{H}_k$ where k indicates the column index of the elements to be driven to zero.

A.1.2 Complex Householder Reflections

Similarly, a Householder reflection can be adapted to be applicable to a vector $\mathbf{x} \in \mathbb{C}^{p\times 1}$ with complex scalar elements by changing equations (A.1) and (A.2) such that

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^H}{\|\mathbf{v}\|_2^2} \tag{A.6}$$

where

$$\mathbf{v} = \mathbf{x} \pm e^{i\arg(x_1)}\|\mathbf{x}\|_2\mathbf{e}_1. \tag{A.7}$$

and $i \triangleq \sqrt{-1}$. This can easily be extended in the same way as the case for matrices with real elements, to be applicable to matrices with complex scalar entries.

A.1.3 Computational Complexity

For the scalar matrix QRD, implementing a Givens rotation approach to the matrix $\mathbf{A} \in \mathbb{R}^{p\times q}$ requires $3p^2(q - p/3)$ flops, whilst the Householder approach requires $2p^2(q - p/3)$ and is therefore computationally less expensive [6]. Note that the computational complexity of the Givens rotations approach can be reduced by implementing multiple rotations at once. However, this approach has not yet been considered for polynomial matrices and is therefore of no relevance here.

A.2 Householder Reflections for Polynomial Matrices

The approach discussed in the previous section can easily be extended to be applicable to polynomial matrices in a very similar way to how Givens rotations were extended to formulate an EPGR[1] in Chapter 5. In order to zero the largest coefficient associated with each element beneath the diagonal of the k^{th} column of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$, firstly the dominant coefficient in each of the $(p - k)$ polynomial elements beneath the diagonal must be located. Again, as with the polynomial matrix decompositions previously discussed in this thesis, the dominant coefficient refers to the coefficient with the largest

[1]EPGR denotes an Elementary Polynomial Givens rotation, which was introduced in Section 5.2.

magnitude within the polynomial element and, if for any polynomial element it is not unique, then any of the dominant coefficients within the element may be chosen. Suppose the set of dominant coefficients beneath the diagonal of the k^{th} column of the matrix are found to be $\{a_{jk}(t_j) : j = k+1, \ldots, p\}$. Once the series of dominant coefficients has been located, a series of elementary delay matrices, previously explained in Section 3.3.2, is then applied to $\underline{\mathbf{A}}(z)$ to obtain the transformed polynomial matrix

$$\underline{\mathbf{A}}^{'}(z) = \underline{\mathbf{B}}^{(p,-t_p)}(z) \ldots \underline{\mathbf{B}}^{(k+2,-t_{k+2})}(z)\underline{\mathbf{B}}^{(k+1,-t_{k+1})}(z)\underline{\mathbf{A}}(z), \tag{A.8}$$

where the $(j,k)^{th}$ polynomial element of this matrix is defined as $\underline{a}'_{jk}(z)$. The objective of this set of elementary delay matrices is to shift each of the dominant coefficients in each polynomial element, beneath the diagonal of the matrix of the k^{th} column, onto the coefficient matrix of order zero, i.e. so that they are positioned on the coefficient matrix $\mathbf{A}(0)$. Then once this has been completed, a Householder reflection matrix $\mathbf{H}_k$ is formulated according to Section A.1 where for equation (A.5) $\mathbf{a}_k = \left[a'_{kk}(0), a'_{(k+1)k}(0), \ldots, a'_{pk}(0)\right]^H$. This matrix is then applied to $\underline{\mathbf{A}}^{'}(z)$ to generate the transformed matrix

$$\underline{\mathbf{A}}^{''}(z) = \mathbf{H}_k\underline{\mathbf{A}}^{'}(z) \tag{A.9}$$

where all coefficients beneath the diagonal in the k^{th} column of the zero-lag coefficient matrix will now equal zero. Furthermore $|a''_{kk}(0)|^2 = \sum_{j=k}^{p} |a'_{jk}(0)|^2 \left(= \|\mathbf{a}_k\|_F^2\right)$. The overall polynomial Householder reflection matrix takes the form

$$\underline{\mathbf{H}}_k(z) = \mathbf{H}(k)\underline{\mathbf{B}}^{(p,-t_p)}(z) \ldots \underline{\mathbf{B}}^{(k+2,-t_{k+2})}(z)\underline{\mathbf{B}}^{(k+1,-t_{k+1})}(z), \tag{A.10}$$

where the subscript k defines the column in which coefficients associated with polynomial elements beneath the diagonal will be driven to zero under application of the polynomial Householder reflection. This polynomial matrix will be referred to as an elementary polynomial Householder reflection (EPHR) and a series of matrices of this form can be applied to a polynomial matrix to transform it to an upper-triangular polynomial matrix.

A.3 Calculation of the PQRD with Polynomial Householder Reflections

Two of the Givens rotation based algorithms, the PQRD-BC and the PQRD-SBR algorithms introduced in Chapter 5, can be modified to apply a series of EPHRs, replacing the EPGRs, to transform a polynomial matrix into an approximately upper-triangular polynomial matrix. The most adaptable of the current PQRD algorithms is the PQRD-BC algorithm for two reasons. Firstly, the PQRD-BC algorithm has been shown, in Chapter 6, to be computationally the most efficient of the three algorithms, typically requiring the least number of iterations to converge. Secondly, the algorithm operates by driving all coefficients associated with polynomial elements beneath the diagonal sufficiently small in each column of the polynomial matrix in turn and so has the appropriate structure to be adapted for EPHRs, which also operate on only a single column of the matrix at any one time. Note that the PQRD-SBR algorithm could also be modified to include EPHRs. However, this algorithm generally requires considerably more iterations to converge that the PQRD-BC algorithm when using EPGRs, due to the polynomial matrix converging to an upper-triangular polynomial matrix through the columns of the matrix from left to right. For this reason, this idea has not been explored further as it would be expected that the same behaviour would be observed. Clearly, if the PQRD-BS algorithm is adapted to include EPHRs, then it will be the same as the modified PQRD-BC algorithm.

A.3.1 The PQRD-BC Algorithm with EPHRs

To adapt the PQRD-BC algorithm to use EPHRs rather than EPGRs, the algorithm will still operate as a series of ordered steps to obtain the decomposition of the polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$

$$\underline{\mathbf{Q}}(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{R}}(z) \tag{A.11}$$

where $\underline{\mathbf{Q}}(z)$ denotes a paraunitary polynomial matrix and $\underline{\mathbf{R}}(z)$ an approximately upper-triangular polynomial matrix.

At each step of the algorithm all coefficients associated with all polynomial elements beneath the diagonal of one column of the polynomial matrix $\underline{\mathbf{A}}(z)$ are driven sufficiently small through an iterative process. At each iteration the dominant coefficient is located within each polynomial element beneath the diagonal in the appropriate column of the matrix. The appropriate EPHR is then calculated according to Section A.2 and applied to the polynomial matrix, resulting in the series of dominant coefficients beneath the diagonal of one column of the polynomial matrix having been driven to zero. This process is repeated until all coefficients associated with polynomial elements beneath the diagonal of the specified column of the polynomial matrix are sufficiently small, i.e. $|a_{jk}(t)| < \epsilon$ where k defines the column index, $j = k+1, \ldots, p$ and $\epsilon > 0$ is some prespecified small value. Once this has been achieved, the algorithm increments the column index by one and therefore repeats the process on the column positioned to the right of the k^{th} column. All other aspects of the PQRD-BC algorithm are the same as those described in Chapter 5 and so multiple sweeps of the algorithm may be required for the algorithm to converge.

The algorithm will generate an approximately upper-triangular polynomial matrix and a proof of convergence for this EPHR variation of the PQRD-BC algorithm will be a very simple modification of the proof of convergence detailed in 5.5.2. The truncation methods discussed in Chapter 4 can again be applied within each iteration of the algorithm to optimise the speed of the algorithm and reduce the orders of the two polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$. However, care must be taken when truncating the polynomial matrices as the transformation performed will no longer be norm-preserving and can therefore result in some error.

A.3.2 Numerical Example: The PQRD-BC Algorithm with Polynomial Householder Reflections

For this example, the modified PQRD-BC algorithm, which implements EPHRs rather than EPGRs, was applied to the fairly simple polynomial matrix $\underline{\mathbf{A}}_2(z)$ previously discussed in Section 6.2.2 of Chapter 6. In Chapter 6 the PQRD-BC algorithm with EPGRs has previously demonstrated good performance when it has been applied to this polynomial matrix and

required 356 iterations to converge to a point where all coefficients associated with polynomial elements beneath the diagonal of the polynomial matrix are less than 10^{-3} in magnitude. For this example, both variations of the PQRD-BC algorithm, i.e. with EPGRs and EPHRs, are again applied to this polynomial matrix. The stopping criterion for these results was set such that each algorithm will stop once all coefficients associated with polynomial elements beneath the diagonal of the polynomial matrix are less than 10^{-2} in magnitude. The energy based truncation method for non para-Hermitian matrices, detailed in Section 4.3.2, was applied to both $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{Q}}(z)$ for these results with $\mu = 10^{-6}$.

The results from applying the PQRD-BC algorithm using both EPHRs and EPGRs are contained in Table A.1. From these results, the EPGR routine can clearly be seen to outperform the Householder approach and this is mainly due to the multiple application of elementary delay matrices before the application of the Householder reflection matrix at each iteration of each step of the algorithm. Initially, it would be expected that this approach will be quite effective. However, once the polynomial matrix becomes more sparse applying multiple delays will make the algorithm computationally slow to implement with many of the coefficients that are now being driven to zero already being sufficiently small, i.e. they are less than the stopping criterion 10^{-2} in magnitude. At this stage a Givens approach would be more appropriate to selectively zero the coefficients that are still larger than this value.

The EPHR approach required considerably more iterations, where each iteration will typically be computationally more expensive than a single iteration of the EPGR variation of the PQRD-BC algorithm. This point is also illustrated by the computational time[2] taken by the two variations of the algorithm to converge, which are also contained in Table A.1 and show the EPGR approach was over six times faster to converge for this example. The EPHR algorithm also produces a paraunitary transformation and an upper-triangular polynomial matrix of very large orders (order of $\underline{\mathbf{Q}}(z)$ is 716 and order of $\underline{\mathbf{R}}(z)$ is 659), which can be seen in Figures A.1 and A.2 respectively. The orders of the polynomial matrices are considerably larger than those obtained using the Givens rotation approach (order of $\underline{\mathbf{Q}}(z)$ is 56 and order of $\underline{\mathbf{R}}(z)$ is 58). Note that the order of both the paraunitary transformation and the upper-

[2]Computations undertaken on a *Intel Centrino Duo* processor with 1GB of RAM.

triangular polynomial matrices could be reduced further by aligning all sets of polynomial coefficients and then truncating again, however, the resulting orders are still found to be considerably larger (order of $\underline{\mathbf{R}}(z)$ reduced to 481 and the order of $\underline{\mathbf{Q}}(z)$ to 538) than those obtained using the EPGR variation of the PQRD-BC algorithm (order of $\underline{\mathbf{R}}(z)$ is 58 and the order of $\underline{\mathbf{Q}}(z)$ to 56). Finally, the Frobenius norm of all elements beneath the diagonal of the final approximately upper-triangular matrix obtained using the EPHR approach was found to be 1.10×10^{-1}, whereas this same measure calculated for the resulting approximately upper-triangular matrix using the EPGR approach was found to be 4.86×10^{-2}. Furthermore, if the stopping condition is reduced so that $\epsilon = 10^{-3}$ and the results are comparable to the results from Section 6.2.2, the PQRD-BC algorithm using EPHRs requires 3231 iterations, taking approximately four minutes to compute the decomposition. The same level of decomposition using the EPGR variation of the algorithm required 362 iterations and took only 0.42 seconds.

To conclude, the EPGR variation of the PQRD-BC algorithm required considerably fewer iterations, and therefore also less time, to converge to an approximately upper-triangular polynomial matrix than the EPHR variation of the algorithm. The resulting matrix from the EPGR variation of the algorithm can also be considered more upper-triangular, as the F-norm of all elements beneath the diagonal of the matrix was considerably smaller. The order of the two matrices obtained by the decomposition was also considerably shorter than those generated by the EPHR modification of the PQRD-BC algorithm, which is an advantage for the potential application of the decomposition to MIMO communication systems discussed in Chapter 8. Finally, the EPGR variation of the algorithm also produced more accurate results with less relative error for this example, due to this variation of algorithm requiring significantly fewer iterations.

	The PQRD-BC Algorithm using	
	Householder Relections	**Givens Rotations**
Number of iterations	658	190
Number of Sweeps	2	2
Final order of $\underline{\mathbf{R}}(z)$	659	58
Final order of $\underline{\mathbf{Q}}(z)$	716	56
E_{rel}	2.58×10^{-2}	9.92×10^{-3}
Final value of g	9.81×10^{-3}	9.67×10^{-3}
Final value of L	1.10×10^{-1}	4.86×10^{-2}
Computational Time (Seconds)	6.63	0.18

Table A.1: Results from applying the PQRD-BC algorithm using Givens rotations and Householder reflections to the polynomial test matrix $\underline{\mathbf{A}}_2(z) \in \mathbb{R}^{3\times3\times4}$.

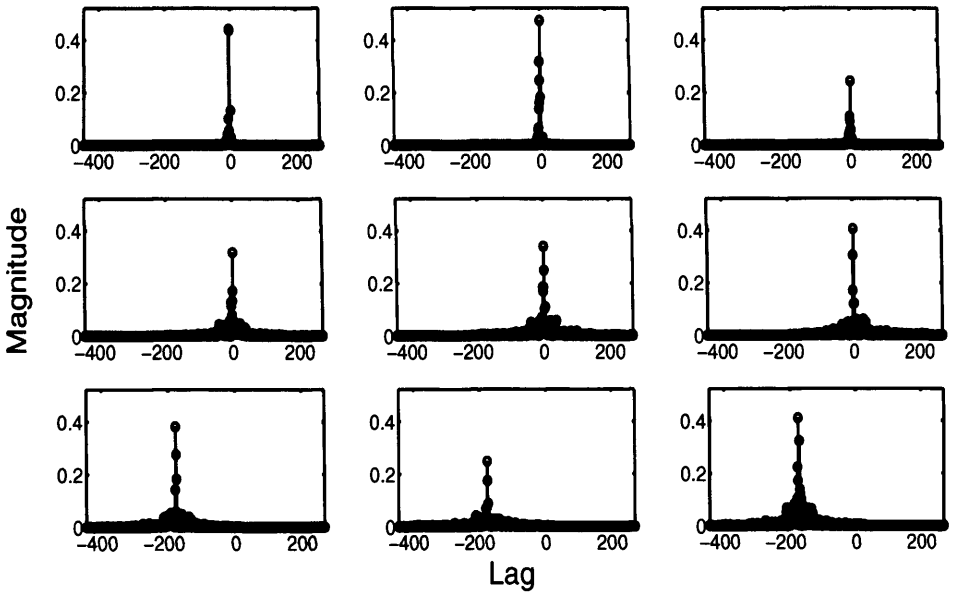

Figure A.1: The coefficients of the polynomial elements of the paraunitary polynomial transformation matrix $\underline{\mathbf{Q}}(z)$ obtained using the PQRD-BC algorithm with polynomial Householder reflections when applied to the polynomial matrix $\underline{\mathbf{A}}_2(z)$.

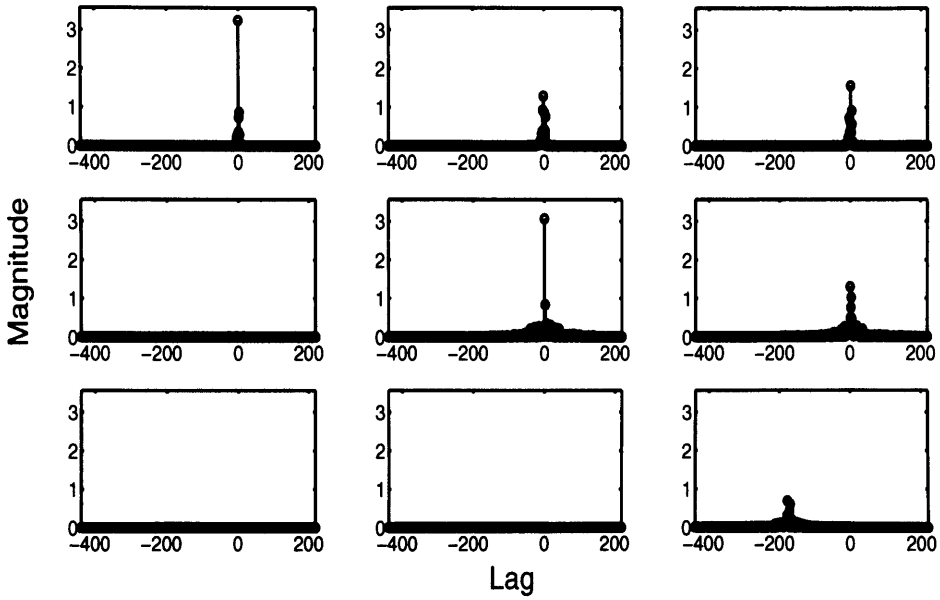

Figure A.2: The coefficients of the polynomial elements of the upper-triangular polynomial matrix $\underline{\mathbf{R}}(z)$ obtained using the PQRD-BC algorithm with polynomial Householder reflections when applied to the polynomial matrix $\underline{\mathbf{A}}_2(z)$.

A.4 Conclusions

This appendix has demonstrated that Householder reflections can be used in conjunction with elementary delay matrices to achieve a polynomial matrix QRD. It has been shown that the PQRD-BC algorithm, detailed in Chapter 5, can easily be modified to include Householder reflections as an alternative to Givens rotations. However, Example A.3.2 confirms that this Householder approach is not as successful as the Givens approach and will require significantly more computational time to converge. Furthermore, the polynomial matrices produced by the Householder approach also have significantly larger orders, which is a considerable disadvantage if the technique is to be used within MIMO communication systems. For this potential application of the decomposition, the order of the polynomial upper-triangular matrix obtained is of critical importance and must be no larger than a certain size for the application to be possible. This point is discussed at length in Chapter 8.

For a scalar matrix decomposition, Householder reflections are extremely useful for driving elements of a matrix to zero on a grand scale. However, Givens rotations are also very useful,

as they provide a much better method for dealing with sparse matrices, where it only required to selectively zero elements of a matrix. However, with polynomial matrices there is an extra dimension to the problem, as each element is now a polynomial and therefore has an associated set of polynomial coefficients, which must all be driven to zero to achieve the decomposition. Unlike the scalar matrix case, the number of Householder transformations required to achieve this cannot be determined in advance. To effectively use polynomial Householder reflections involves multiple applications of elementary delay matrices in succession and, from the results presented here, this appears to cause the coefficients to disperse over an unnecessarily large numbers of coefficient lags of the matrix. Following a small number of iterations this will create a sparse polynomial matrix of a very large order, where the Givens method is a far more suitable technique of achieving the decomposition. The result is an algorithm that is computationally slow to implement due to the unnecessarily large orders of the polynomial matrices. This is not a problem in the original PQRD-BC algorithm, which operates by applying EPGRs.

Further research could be undertaken to investigate the concept of using EPHRs as an alternative to EPGRs and a more comprehensive study of the different variations of the algorithm for a range of polynomial input matrices would be beneficial. In particular, it would be interesting to see if there is any advantage of implementing an algorithm combining both approaches, i.e. initially implement the EPHR variation of the algorithm for the first few iterations of each step, (whilst the matrix is still not sparse) and then implement the original PQRD-BC algorithm using EPGRs once a significant number of the coefficients associated with the polynomial elements in the particular column are less than the specified stopping criterion.

Appendix B

Summary of the Decomposition Algorithms

A summary of each of the algorithms used within this thesis for calculating the PQRD and PSVD are given here. A summary of the SBR2 algorithm has not been included, as a detailed summary can be found in [7].

B.1 Summary of the PSVD Techniques

B.1.1 The PSVD by PEVD Algorithm

The PSVD by PEVD Algorithm

1: **Input** polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$ to be factorised
2: **Specify** the convergence parameter ϵ, the truncation parameter μ and the maximum number of iterations of the algorithm MaxIter
3: Calculate the matrix $\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z)$.
4: Apply the SBR2 algorithm to this matrix to calculate the paraunitary matrix $\underline{\mathbf{U}}(z)$.
5: Calculate the matrix $\widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z)$.
6: Apply the SBR2 algorithm to this matrix to calculate the paraunitary matrix $\underline{\mathbf{V}}(z)$.
7: Calculate the diagonal matrix $\underline{\mathbf{D}}(z) = \underline{\mathbf{U}}(z)\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{V}}}(z)$.

B.1.2 The PSVD by PQRD Algorithm

The PSVD by PQRD Algorithm

1: **Input** polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$ to be factorised
2: **Specify** the convergence parameter ϵ, the truncation parameter μ and the maximum number of iterations of the algorithm MaxIter
3: Set $\underline{\mathbf{U}}(z) = \mathbf{I}_p$ and $\underline{\mathbf{V}}(z) = \mathbf{I}_q$
4: Set iter $= 0$ and $g = 1 + \epsilon$
5: **while** iter $<$ MaxIter and $g > \epsilon$ **do**
6: Find indices j, k and t where $j \neq k$ such that $|a_{jk}(t)| \geq |a_{mn}(\tau)|$ holds for $m = 1, \ldots, p$, $n = 1, \ldots, q$ such that $m \neq n$ and $\forall \tau \in \mathbb{Z}$
7: Set $g = |a_{jk}(t)|$
8: **if** $g > \epsilon$ **then**
9: iter $\Leftarrow$ iter $+ 1$
10: Apply any of the three algorithms for calculating the PQRD to $\underline{\mathbf{A}}(z)$ to obtain the decomposition $\underline{\mathbf{U}}_1(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{A}}'(z)$, where $\underline{\mathbf{A}}'(z)$ is an approximately upper triangular polynomial matrix.
11: Set $\underline{\mathbf{A}}''(z) = \widetilde{\underline{\mathbf{A}}}'(z)$
12: $\underline{\mathbf{U}}(z) \Leftarrow \underline{\mathbf{U}}_1(z)\underline{\mathbf{U}}(z)$
13: Apply any of the three algorithms for calculating the PQRD to $\underline{\mathbf{A}}''(z)$ to obtain the decomposition $\underline{\mathbf{V}}_1(z)\underline{\mathbf{A}}''(z) = \underline{\mathbf{A}}'''(z)$, where $\underline{\mathbf{A}}'''(z)$ is an approximately upper triangular polynomial matrix.
14: $\underline{\mathbf{V}}(z) \Leftarrow \underline{\mathbf{V}}_1(z)\underline{\mathbf{V}}(z)$
15: Set $\underline{\mathbf{A}}(z) = \widetilde{\underline{\mathbf{A}}}'''(z)$.
16: Apply the energy based truncation method detailed in Section 4.3 to the polynomial matrices $\underline{\mathbf{A}}(z)$, $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$
17: **end if**
18: **end while**
19: Set $\underline{\mathbf{S}}(z) = \underline{\mathbf{A}}(z)$.
20: The overall decomposition performed is $\underline{\mathbf{U}}(z)\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{V}}}(z) = \underline{\mathbf{S}}(z)$.

B.2 Summary of the Algorithms for Calculating the PQRD

B.2.1 The PQRD By Steps Algorithm

The PQRD-BS Algorithm

Input polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ to be factorised
Specify the convergence parameter ϵ, the truncation parameter μ and the maximum number of iterations per step of the algorithm MaxIter
Set $\underline{\mathbf{Q}}(z) = \mathbf{I}_p$
for $j = 2, \ldots, p$ **do**
  for $k = 1, \ldots, \min\{j-1, q\}$ **do**
    Set iter $= 0$ and $g = 1 + \epsilon$
    while iter $<$ MaxIter and $g > \epsilon$ **do**
      Find lag index t such that $|a_{jk}(t)| \geq |a_{jk}(\tau)|$ holds $\forall \tau \in \mathbb{Z}$
      Set $g = |a_{jk}(t)|$
      if $g > \epsilon$ **then**
        iter $\Leftarrow$ iter $+ 1$
        Calculate the rotation angles θ, α and ϕ according to equations (5.5), (5.6) and (5.7)
        Calculate the EPGR $\underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ according to Section 5.2
        $\underline{\mathbf{A}}(z) \Leftarrow \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z)$
        $\underline{\mathbf{Q}}(z) \Leftarrow \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{Q}}(z)$
        Apply the energy based truncation method detailed in Section 4.3 to the polynomial matrices $\underline{\mathbf{A}}(z)$ and $\underline{\mathbf{Q}}(z)$
      end if
    end while
  end for
end for
If the coefficient with largest magnitude associated with any of the elements beneath the diagonal is $> \epsilon$ then a second sweep of the algorithm is required and so the process is repeated from step 4.
Once all sweeps have been completed, set $\underline{\mathbf{R}}(z) = \underline{\mathbf{A}}(z)$. The decomposition performed is of the form $\underline{\mathbf{Q}}(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{R}}(z)$.

B.2.2 The PQRD By Columns Algorithm

The PQRD-BC Algorithm

Input polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p\times q}$ to be factorised
Specify the convergence parameter ϵ, the truncation parameter μ and the maximum number of iterations per step of the algorithm MaxIter
Set $\underline{\mathbf{Q}}(z) = \mathbf{I}_p$
for $k = 1, \ldots, \min\{p-1, q\}$ **do**
  Set iter $= 0$ and $g = 1 + \epsilon$
  while iter $<$ MaxIter and $g > \epsilon$ **do**
    Find indices j and t such that $|a_{jk}(t)| \geq |a_{mk}(\tau)|$ holds for $m = (k+1), \ldots, p$ and $\forall \tau \in \mathbb{Z}$
    Set $g = |a_{jk}(t)|$
    if $g > \epsilon$ **then**
      iter $\Leftarrow$ iter $+ 1$
      Calculate the rotation angles θ, α and ϕ according to equations (5.5), (5.6) and (5.7)
      Calculate the EPGR $\widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ according to Section 5.2
      $\underline{\mathbf{A}}(z) \Leftarrow \widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z)$
      $\underline{\mathbf{Q}}(z) \Leftarrow \widehat{\underline{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{Q}}(z)$
      Apply the energy based truncation method detailed in Section 4.3 to the polynomial matrices $\underline{\mathbf{A}}(z)$ and $\underline{\mathbf{Q}}(z)$
    end if
  end while
end for
If the coefficient with largest magnitude associated with any of the elements beneath the diagonal is $> \epsilon$ then a second sweep of the algorithm is required and so the process is repeated from step 4.

B.2.3 The PQRD By Sequential Best Rotation Algorithm

The PQRD-SBR Algorithm

Input polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{\mathbb{C}}^{p \times q}$ to be factorised
Specify the convergence parameter ϵ, the truncation parameter μ and the maximum number of iterations of the algorithm MaxIter
Set $\underline{\mathbf{Q}}(z) = \mathbf{I}_p$, iter $= 0$ and $g = 1 + \epsilon$
while iter $<$ MaxIter and $g > \epsilon$ **do**
  Find indices j, k and t where $j > k$ such that $|a_{jk}(t)| \geq |a_{mn}(\tau)|$ holds for $m = 2, \ldots, p$, $n = 1, \ldots, q$ such that $m > n$ and $\forall \tau \in \mathbb{Z}$
  Set $g = |a_{jk}(t)|$
  if $g > \epsilon$ **then**
    iter $\Leftarrow$ iter $+ 1$
    Calculate the rotation angles θ, α and ϕ required to drive g to zero according to equations (5.5), (5.6) and (5.7)
    Calculate the EPGR $\underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)$ according to Section 5.2
    $\underline{\mathbf{A}}(z) \Leftarrow \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{A}}(z)$
    $\underline{\mathbf{Q}}(z) \Leftarrow \underline{\widehat{\mathbf{G}}}^{(j,k,\alpha,\theta,\phi,t)}(z)\underline{\mathbf{Q}}(z)$
    Calculate the inverse time-shift matrix $\underline{\mathbf{B}}^{(k,t)}(z)$ according to equation 5.30.
    $\underline{\mathbf{A}}(z) \Leftarrow \underline{\mathbf{B}}^{(k,t)}(z)\underline{\mathbf{A}}(z)$
    $\underline{\mathbf{Q}}(z) \Leftarrow \underline{\mathbf{B}}^{(k,t)}(z)\underline{\mathbf{Q}}(z)$
    Apply the energy based truncation method detailed in Section 4.3 to the polynomial matrices $\underline{\mathbf{A}}(z)$ and $\underline{\mathbf{Q}}(z)$
  end if
end while

Appendix C

Computational Complexity of the Polynomial Matrix Decompositions

The computational complexity of each of the polynomial matrix decomposition algorithm is calculated by counting the number of multiplication, division, addition and subtraction operations throughout each iteration. The results are demonstrated in the series of tables given below.

C.1 The SBR2 Algorithm

Suppose the input to the SBR2 algorithm is the polynomial matrix $\underline{\mathbf{R}}(z) \in \mathbb{C}^{p\times p\times T}$. The application of the elementary delay matrices at each iteration of the algorithm will affect the order of the matrix. Suppose at the end of the i^{th} iteration of the algorithm, the para-Hermitian polynomial updated matrix $\underline{\mathbf{D}}_i(z)$ is of order T_i. Without any operations to truncate the orders of the polynomial matrices within the algorithm, the value of this parameter will increase at each iteration by $2\,|t|$, where t defines the lag index of the dominant coefficient which has been driven to zero at that iteration. However, it is not so simple when a truncation method is used.

Table C.1 gives a detailed break down of the computational complexity and storage requirements within one iteration of the SBR2 algorithm when in its most simplistic form, before the addition of any functions to limit the growing order of the polynomial matrices or stopping criterion.

	Complexity	Storage	Other Comments
Find dominant coefficient	$p^2(T_{i-1}+1)$	-	$p^2T_i - 1$ comparisons
Increment iteration	1	-	-
Calculate the Shift Parameter	3	-	-
Apply left and right delay to $\underline{\mathbf{R}}(z)$	-	Increase order of $\underline{\mathbf{D}}_{i-1}(z)$ by $2\lvert t\rvert$	-
Calculate Rotation Angles	4	-	1 trigonometric function
Apply left rotate and right rotations to $\underline{\mathbf{R}}(z)$	$4(3p(T_i+1)+1)$	-	2 trigonometric functions are used

Table C.1: Computational Complexity of the SBR2 Algorithm - This does not include computation of the paraunitary transformation matrix $\underline{\mathbf{H}}(z)$ or allow for any truncation methods.

The computational complexity demonstrated in Table C.1 could be reduced further if the para-Hermitian nature of the polynomial matrix $\underline{\mathbf{D}}_i(z)$ was exploited. For example, the complexity for the rotations could be reduced to $3(p(T_i+1)+1)+4$. The amount of elements to be stored at each iteration could also be reduced by exploiting this property. Rather than storing p^2T_i elements, only $\frac{p}{2}(T_i+p+1)$ need to be stored. Accordingly the amount of operations and comparisons required to find the dominant coefficient will also be reduced.

C.1.1 Implementing a Truncation Method

It is difficult to comment on how the truncation method has reduced the computational complexity of the SBR2 algorithm as it is entirely dependent on the order of the transformed matrix at each iterations, which can no longer be predetermined. However, in general the computational complexity is significantly reduced using a truncation method, despite the fact that further computations are required to implement the methods at each iteration.

C.2 Algorithms for Calculating the PQRD

C.2.1 The PQRD by Steps Algorithm

Suppose at the start of iteration i, the matrix $\underline{\mathbf{A}}(z)$ has dimenions $p \times q$ and is of order T_1. Furthermore, suppose that the dominant coefficient at this iterations is found to be $a_{jk}(t)$. The computational compexity, of the first algorithm, over this iteration is summarised below in Table C.2.1. This table does not include any computations for updating the paraunitary matrix $\underline{\mathbf{Q}}(z)$, as this is need not be calculated within the main iterative routine of the algorithm. It has been assumed that the zerolag plane has been specified prior to beginning each iteration.

	Complexity	**Storage**	**Other Comments**
Calculating the indices	1	-	1 comparison
Find dominant coefficient in (and find lag index) $\underline{a}_{jk}(z)$	$T_1 + 1$	-	$T - 1$ comparisons
Increment iteration	1	-	-
Apply Left Delay to $\underline{\mathbf{A}}(z)$	-	Increase order of $\underline{\mathbf{A}}(z)$ by $\|t\|$	-
Keeping track of the zerolag plane of $\underline{\mathbf{A}}(z)$	1	-	-
Calculate Rotation Angles	2	-	1 trigonometric function
Apply left rotate to $\underline{\mathbf{A}}(z)$	$3q\,(T_1 + \|t\|) + 4$	-	2 trig functions
Truncating $\underline{\mathbf{A}}(z)$	$pT_1\,(2q-1)$ + $(c_L + c_R)\,(2pq - p) + 4$	Reduces the order of $\underline{\mathbf{A}}(z)$ if it is unnecessarily large	-

Table C.2: Table demonstrating the computational complexity of the PQRD-BS algorithm for calculating the upper-triangular matrix $\underline{\mathbf{R}}(z)$ and comments on the storage requirements of this algorithm.

C.2.2 The PQRD by Columns Algorithm

Again assume at the start of iteration i, the matrix $\underline{\mathbf{A}}(z)$ has dimenions $p \times q$ and is of order T_1, with the dominant coefficient at this iterations found to be $a_{jk}(t)$. The computational complexity over this iteration for updating the polynomial matrices $\underline{\mathbf{A}}(z)$ can be seen in Table C.2.2.

	Complexity	**Storage**	**Other Comments**
Calculate indices	1	-	-
Find dominant coefficient (and find indices) in $\underline{a}_{jk}(z)$	$2pT_1 + 1$	-	$pqT - 1$ comparisons
Increment iteration	1	-	-
Apply Left Delay to $\underline{\mathbf{A}}(z)$	-	Increase order of $\underline{\mathbf{A}}(z)$ by $\|t\|$	-
Keeping track of the zerolag plane of $\underline{\mathbf{A}}(z)$	1	-	-
Calculate Rotation Angles	2	-	1 trigonometric function
Apply left rotate to $\underline{\mathbf{A}}(z)$	$3q\,(T_1 + \|t\|) + 4$	-	2 trigonometric functions
Truncating $\underline{\mathbf{A}}(z)$	$pT_1\,(2q-1)$ + $(c_L + c_R)\,(2pq - p) + 4$	Reduces the order of $\underline{\mathbf{A}}(z)$ if it is unnecessarily large	-

Table C.3: Table demonstrating the computational complexity of this algorithm for calculating the upper-triangular matrix $\underline{\mathbf{R}}(z)$ and comments on the storage requirements of the algorithm.

C.2.3 The PQRD by SBR Algorithm

Assume again that the polynomial matrices $\underline{\mathbf{A}}(z)$ at the start of iteration i is of order T_1. Furthermore, assume that the dominant coefficient at this iteration is found to be $a_{jk}(t)$. The computational complexity over a single iteration of the PQRD-SBR algorithm can be found in Table C.4.

	Complexity	**Storage**	**Other Comments**
Find dominant coefficient (and find indices)	$2pqT+1$	-	$pqT-1$ comparisons
Increment iteration	1	-	-
Apply Left Delay to $\underline{\mathbf{A}}(z)$	-	Increase order of $\underline{\mathbf{A}}(z)$ by $\lvert t\rvert$	-
Keeping track of the zerolag plane of $\underline{\mathbf{A}}(z)$	1	-	-
Calculate Rotation Angles	2	-	1 trigonometric function
Apply left rotate to $\underline{\mathbf{A}}(z)$	$3q\left(T+\lvert t\rvert\right)+4$	-	2 trigonometric functions
Apply undelay	1	Order again increases by $\lvert t\rvert$	-
Truncating $\underline{\mathbf{A}}(z)$	$pT_A\left(2q-1\right)+\left(c_L+c_R\right)\left(2pq-p\right)+4$	Reduces the order of $\underline{\mathbf{A}}(z)$ if it is unnecessarily large	-

Table C.4: Table demonstrating the computational complexity of this algorithm for calculating the upper-triangular matrix $\underline{\mathbf{R}}(z)$ and comments on the storage requirements of the algorithm.

Appendix D

Illustrations for Chapter 8

D.1 Polynomial Channel Matrices

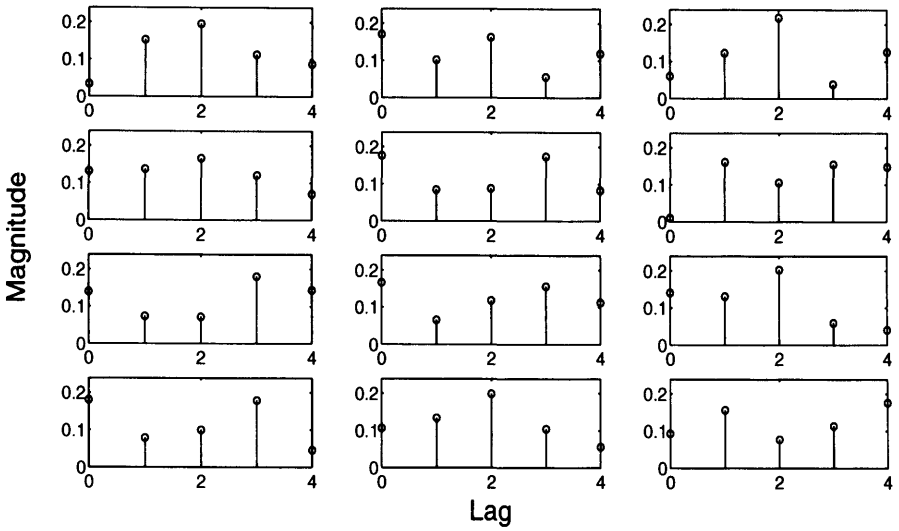

Figure D.1: A stem plot representation of the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

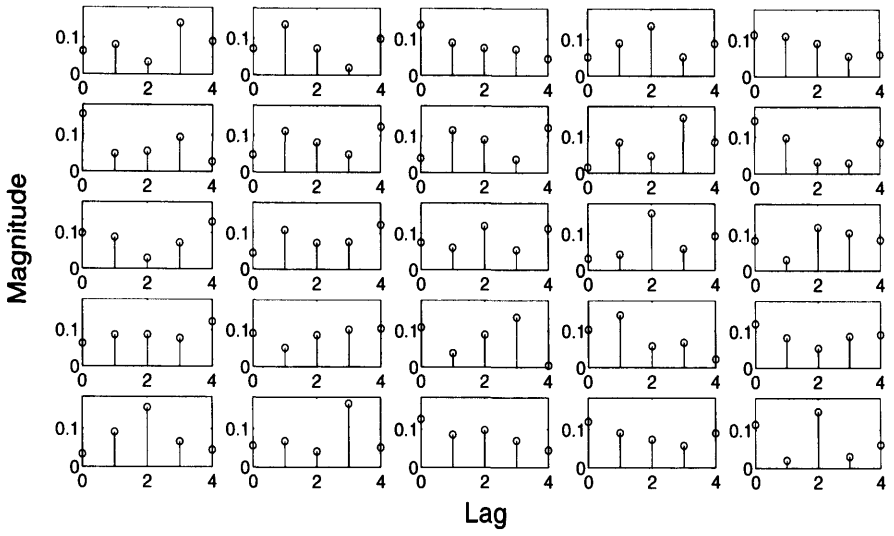

Figure D.2: A stem plot representation of the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.

D.2 PQRD

D.2.1 Channel Matrix 1

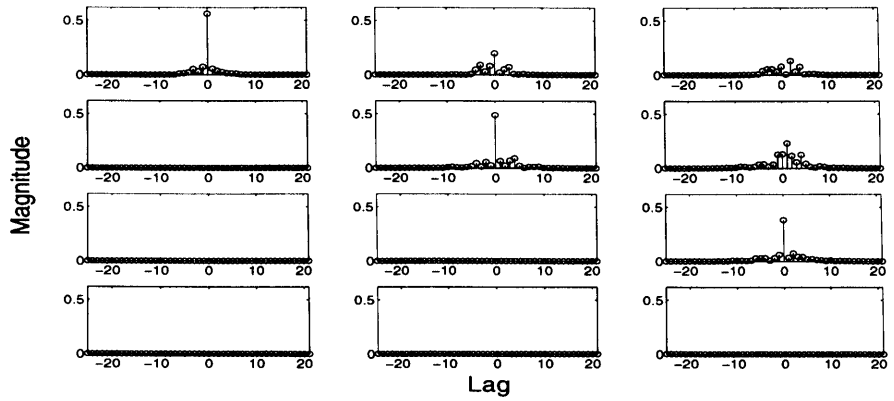

Figure D.3: The coefficients of the polynomial elements of the approximately upper-triangular polynomial matrix $\underline{\mathbf{R}}(z)$ obtained by applying the PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

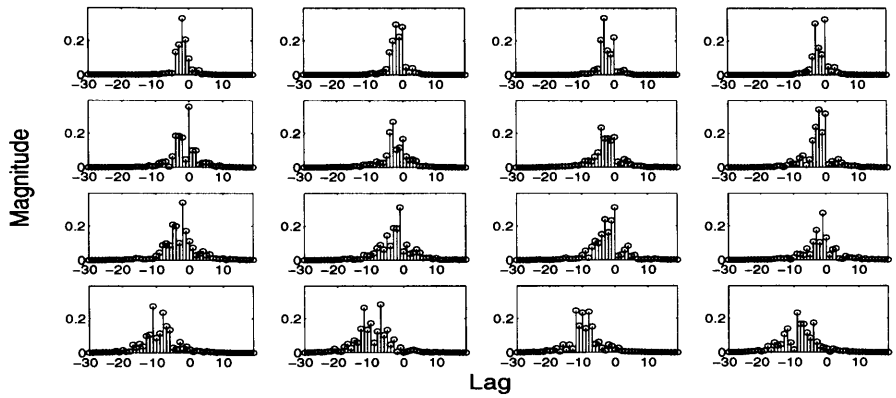

Figure D.4: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{Q}}(z)$ obtained by applying the PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

D.2.2 Channel Matrix 2

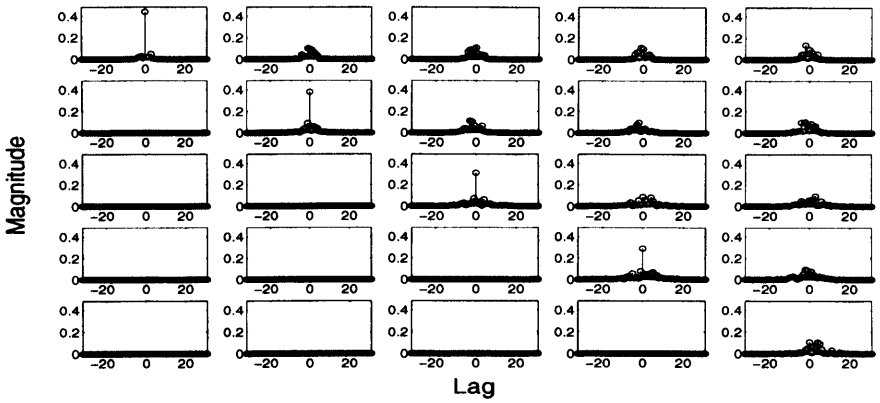

Figure D.5: The coefficients of the polynomial elements of the approximately upper-triangular polynomial matrix $\underline{\mathbf{R}}(z)$ obtained by applying the PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.

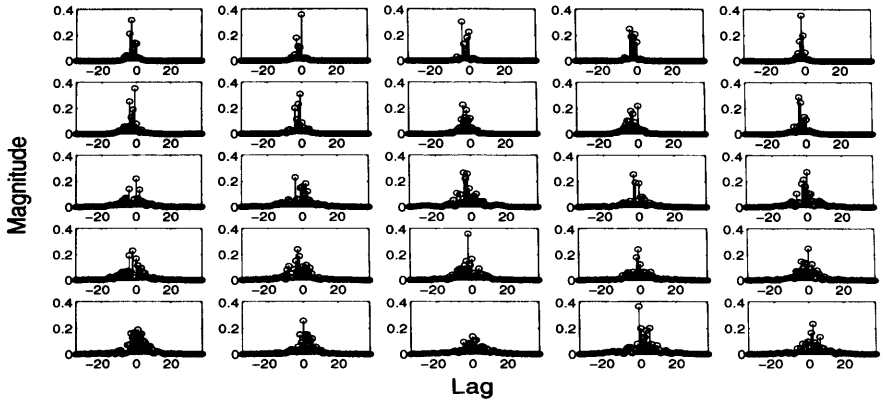

Figure D.6: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{Q}}(z)$ obtained by applying the PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.

D.3 PSVD

D.3.1 Channel Matrix 1

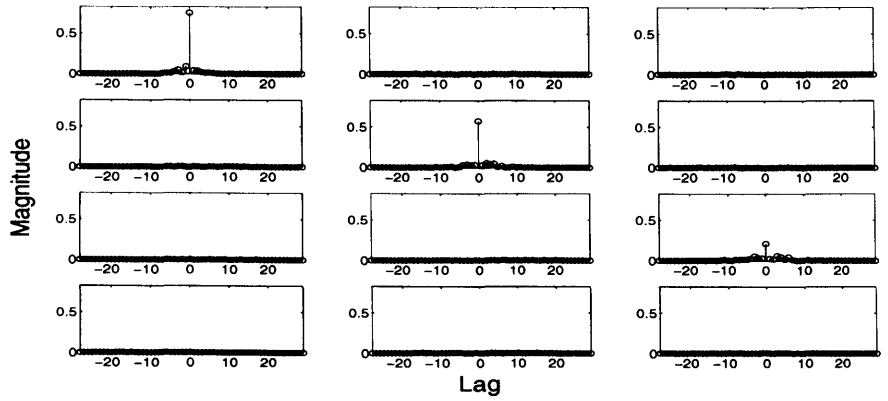

Figure D.7: The coefficients of the polynomial elements of the approximately diagonal polynomial matrix $\underline{\mathbf{S}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

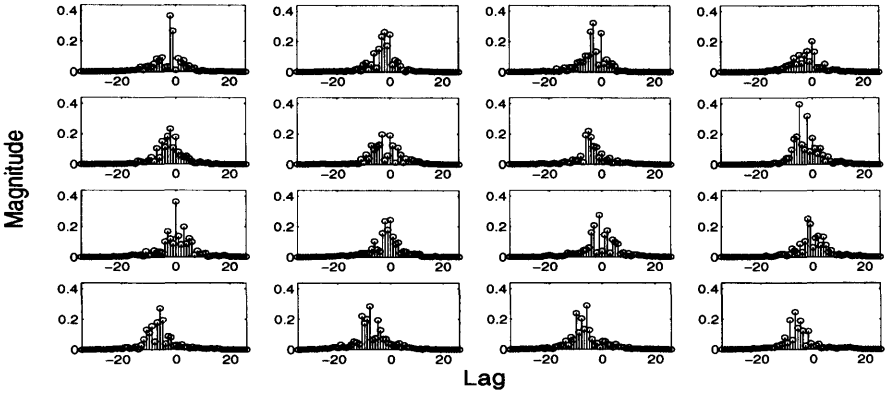

Figure D.8: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{U}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

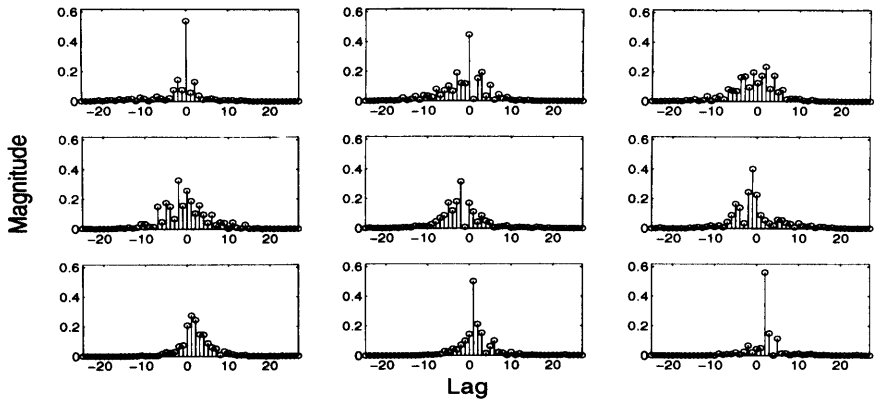

Figure D.9: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{V}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_1(z)$.

D.3.2 Channel Matrix 2

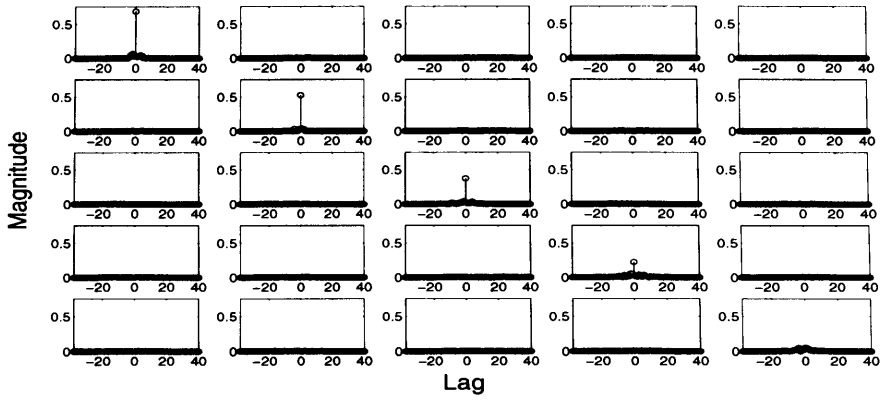

Figure D.10: The coefficients of the polynomial elements of the approximately diagonal polynomial matrix $\underline{\mathbf{S}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.

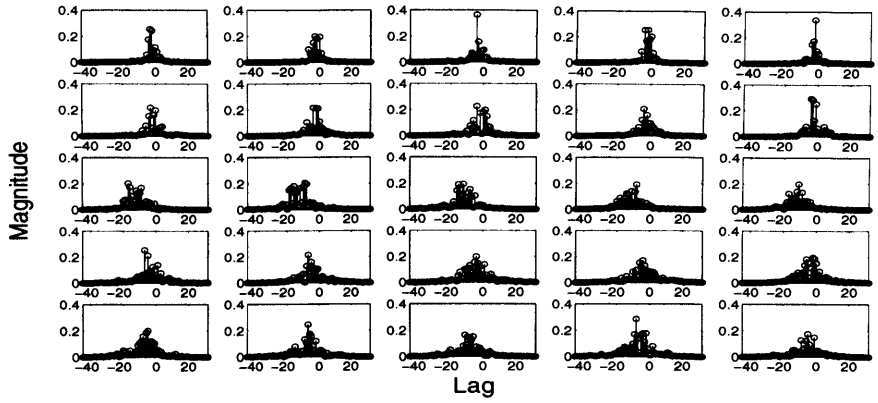

Figure D.11: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{U}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.

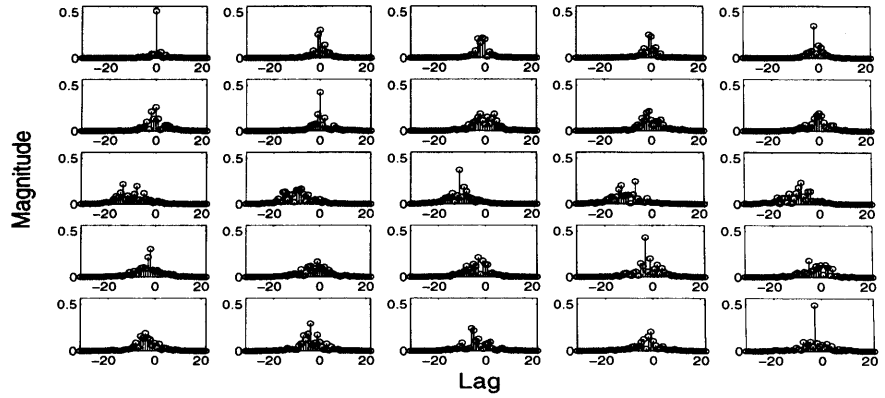

Figure D.12: The coefficients of the polynomial elements of the paraunitary polynomial matrix $\underline{\mathbf{V}}(z)$ obtained by applying the PSVD by PQRD-BC algorithm to the polynomial channel matrix $\underline{\mathbf{C}}_2(z)$.