AN ALGORITHM FOR POLYNOMIAL MATRIX SVD BASED ON GENERALISED KOGBETLIANTZ TRANSFORMATIONS

John G McWhirter

School of Engineering, Cardiff University
Queen's Buildings, CF24 3AA, Cardiff, Wales
phone: + (44) (0)2920870627, fax: + (44) (0)2920874716, email: mcwhirterjg@cardiff.ac.uk

ABSTRACT

An algorithm is presented for computing the singular value decomposition (SVD) of a polynomial matrix. It takes the form of a sequential best rotation (SBR) algorithm and constitutes a generalisation of the Kogbetliantz technique for computing the SVD of conventional scalar matrices. It avoids "squaring" the matrix to be factorised, uses only unitary and paraunitary operations, and therefore exhibits a high degree of numerical stability.

1. INTRODUCTION

Polynomial matrices have been used for many years in the area of control [1]. They play an important role in the realisation of multi-variable transfer functions associated with multiple-input multiple-output (MIMO) systems. Over the last few years they have become more widely used in the context of digital signal processing (DSP) and communications [2]. Typical areas of application include broadband adaptive sensor array processing [3, 4], MIMO communication channels [5–7], and digital filter banks for subband coding [8] or data compression [9].

Just as orthogonal or unitary matrix decomposition techniques such as the QR decomposition (QRD), eigenvalue decomposition (EVD), and singular value decomposition (SVD) [10] are important for narrowband adaptive sensor arrays [11], corresponding paraunitary polynomial matrix decompositions are proving beneficial for broadband adaptive arrays [12–14] and also for filterbank design [15, 16]. In a previous paper [17], we described a generalisation of the EVD for conventional Hermitian matrices to para-Hermitian polynomial matrices. This technique will be referred to as PEVD while the underlying algorithm is known as the 2nd order sequential best rotation (SBR2) algorithm.

In order to minimise the number of iterative steps required, and so prevent unnecessary growth in the order of the polynomial matrix being diagonalised, the philosophy adopted for the algorithm was one of sequential best rotation (SBR). In the context of conventional Hermitian matrices, this corresponds to the classical Jacobi algorithm [18]. The SBR2 algorithm is, in effect, a generalisation of the classical Jacobi algorithm to the third (time) dimension associated with polynomial matrices. A similar approach was subsequently adopted for polynomial matrix QR decomposition (PQRD) [19].

For similar reasons, the PSVD algorithm outlined here is also based on the SBR principle. It can be viewed as the generalisation to polynomial matrices of an SBR algorithm for computing the SVD of conventional matrices. Unlike the case of Hermitian matrix EVD, no such algorithm seems to exist already in the literature. This is not surprising since, for conventional matrices, basing an algorithm entirely on the SBR philosophy would not be computationally efficient. Given a matrix $\mathbf{X} \in \mathbb{C}^{m\times n}$, where we assume that $m > n$, it is more efficient to begin the SVD by performing a QR decomposition of the matrix since this only requires a fixed number of carefully structured steps. In this case, a unitary matrix $\mathbf{Q} \in \mathbb{C}^{m\times m}$ is computed s.t.

$$\mathbf{QX} = \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \tag{1}$$

where $\mathbf{R} \in \mathbb{C}^{n\times n}$ is an upper triangular matrix with real diagonal elements. Computation of the SVD can then proceed by treating $\mathbf{R}$ as a general square matrix and performing an iterative sequence of Kogbetliantz transformations which is guaranteed to converge and reduce it to diagonal form. The Kogbetliantz transformation may be viewed as a generalisation of the classical Jacobi algorithm to non-symmetric (square) matrices and also belongs to the class of SBR algorithms [18]. It results in a transformation of the form

$$\mathbf{Q}_1\mathbf{RV} = \mathbf{\Sigma} \tag{2}$$

where $\mathbf{Q}_1 \in \mathbb{C}^{n\times n}$ and $\mathbf{V} \in \mathbb{C}^{n\times n}$ are unitary matrices and $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$ is diagonal. In combination, we have

$$\mathbf{UXV} = \begin{bmatrix} \mathbf{\Sigma} \\ 0 \end{bmatrix} \tag{3}$$

where $\mathbf{U} \in \mathbb{C}^{m\times m}$ is a unitary matrix given by

$$\mathbf{U} = \begin{bmatrix} \mathbf{Q}_1 & 0 \\ 0 & \mathbf{I} \end{bmatrix}\mathbf{Q} \tag{4}$$

and so this constitutes the SVD of $\mathbf{X}$ [10].

The first stage in developing a PSVD algorithm based on the SBR philosophy, is to generate an SBR algorithm for the case of conventional matrices. In effect, it is necessary to merge the QR decomposition stage of the SVD algorithm described above, into the iterative Kogbetliantz process. A novel algorithm of this type is developed in the next section for complex (scalar) matrices. Note that the complex case is more involved than its real counterpart because of the need to ensure that the diagonal elements remain real throughout the process. This is vital for the Jacobi transformation step which relies on Hermitian symmetry of the 2×2 sub-matrix to which it is applied.

2. MODIFIED KOGBETLIANTZ ALGORITHM FOR COMPLEX NON-SQUARE MATRICES

Assuming that $m > n$, we seek to compute the SVD of a matrix $\mathbf{X} \in \mathbb{C}^{m\times n}$ as defined by equation (3). Note that for square matrices, it is not necessary to perform the initial QR decomposition as indicated in equation (1) so the basic Kogbetliantz method is sufficient. The corresponding PSVD algorithm can easily be deduced from the one developed below and will not be treated separately in this paper.

2.1 Initial phase adjustment

The algorithm begins by transforming the matrix to one with real elements on the principal diagonal. This can be achieved by multiplying the jth column by $\exp(-i\alpha_j)$ where α_j is the phase of x_{jj} i.e. $x_{jj} = |x_{jj}|\exp(i\alpha_j)$. The same procedure is applied to every column and so the entire process may be written in the form

$$\mathbf{X} \leftarrow \mathbf{XT} \tag{5}$$

where $\mathbf{T} \in \mathbb{C}^{n\times n}$ is a diagonal matrix of phase rotations and is therefore unitary. Note that $\|\mathbf{X}\|$ is not affected by this initial phase adjustment (where $\|.\|$ denotes the Frobenius norm of a matrix or of a polynomial matrix as defined in [17]). The algorithm ensures that each subsequent transformation maintains the real property of the diagonal elements of the matrix. This occurs naturally with the Jacobi transformation which preserves Hermitian symmetry, but requires more effort in other cases.

The new SBR algorithm begins by locating the dominant off-diagonal element of $\mathbf{X}$ (i.e. the one with greatest magnitude) denoted by x_{jk}. The next step depends on whether or not $j > n$.

2.2 Givens rotation

If $j > n$ (i.e. if the dominant off-diagonal element lies outside the upper $n \times n$ sub-matrix), compute and apply a complex Givens rotation with rotation parameters defined by

$$\begin{bmatrix} c & se^{i\phi} \\ -se^{-i\phi} & c \end{bmatrix} \begin{bmatrix} x_{kk} \\ x_{jk} \end{bmatrix} = \begin{bmatrix} x'_{kk} \\ 0 \end{bmatrix} \tag{6}$$

where $c = \cos\theta$, $s = \sin\theta$, and $x_{kk}, x'_{kk} \in \mathbb{R}$. This requires

$$-se^{-i\phi}x_{kk} + cx_{jk} = 0 \tag{7}$$

while

$$x'_{kk} = cx_{kk} + se^{i\phi}x_{jk} \tag{8}$$

Denoting $x_{jk} = |x_{jk}|\, e^{i\omega}$, it is clear that we must have $\phi = -\omega$ to ensure that x'_{kk} is real. Equation (7) then becomes

$$-se^{i\omega}x_{kk} + c\,|x_{jk}|\, e^{i\omega} = 0 \tag{9}$$

which is satisfied by

$$\tan\theta = |x_{jk}|\,/x_{kk} \tag{10}$$

We denote this transformation by

$$\mathbf{X}' = \mathbf{G}(\theta, \phi)\mathbf{X} \tag{11}$$

where $\mathbf{X}' \in \mathbb{C}^{m\times n}$ and $\mathbf{G}(\theta, \phi) \in \mathbb{C}^{m\times m}$ takes the form of a suitably embedded Givens rotation [10] - i.e. an $m \times m$ unit matrix except for the $(k,k), (k, j), (j,k)$ and (j, j) elements which serve to embed the 2×2 rotation matrix in equation (6). This completes the elementary transformation for the case $j > n$. In preparation for the next iteration the input matrix is now updated by setting $\mathbf{X} \leftarrow \mathbf{X}'$. Note that $\|\mathbf{X}'\| = \|\mathbf{X}\|$ and that for each iteration, $\|\text{diag}(\mathbf{X})\|^2$ (the on-diagonal energy) increases while $\|\text{offdiag}(\mathbf{X})\|^2$ (the off-diagonal energy) decreases by the same amount. The operator $\mathbf{G}(\theta, \phi)$ in equation(11) contributes to the overall transformation $\mathbf{U}$ in equation (3).

2.3 Complex Kogbetliantz transformation

If $j \leq n$ (i.e. if the dominant off-diagonal element of $\mathbf{X}$ lies within the upper $n \times n$ sub-matrix), perform the following complex Kogbetliantz transformation designed to eliminate both x_{jk} and x_{kj} [18]. This transformation may be broken down into three simple steps - a Givens rotation and a symmetrisation followed by a standard Jacobi transformation. The Givens rotation step, which would not be required in the real case, is needed here to ensure that the symmetrisation step leaves the diagonal elements entirely real. First, swap the indices j and k, if necessary, so that $j > k$. It is advisable also to swap rows j and k, and columns j and k, so that the Givens rotation which follows, is never used to eliminate a zero element and thus become numerically ill-defined.

2.3.1 Givens rotation

Compute and apply a Givens rotation of the form

$$\begin{bmatrix} c & se^{i\phi} \\ -se^{-i\phi} & c \end{bmatrix} \begin{bmatrix} x_{kk} & x_{kj} \\ x_{jk} & x_{jj} \end{bmatrix} = \begin{bmatrix} x'_{kk} & x'_{kj} \\ 0 & x'_{jj} \end{bmatrix} \tag{12}$$

where x_{jj}, x_{kk} and $x'_{kk} \in \mathbb{R}$. This requires the same choice of rotation parameters as the Givens rotation applied if $j > n$. However we also have

$$x'_{jj} = -se^{i\omega}x_{kj} + cx_{jj} \tag{13}$$

This quantity is not necessarily real, so a further phase adjustment is required. Denote $x'_{jj} = \left|x'_{jj}\right| e^{i\beta}$ and multiply the jth row of the matrix by $e^{-i\beta}$. This ensures that the matrix still has real elements on the diagonal. Denote the embedded Givens rotation, together with any phase adjusment and interchange of rows and columns, by

$$\mathbf{X}' = \mathbf{F}(\theta, \phi, \beta)\mathbf{X} \tag{14}$$

where $\mathbf{F}(\theta, \phi, \beta) \in \mathbb{C}^{m\times m}$. Once again, since this combined transformation is unitary, it can be seen that $\|\mathbf{X}'\| = \|\mathbf{X}\|$ and that $\|\text{diag}(\mathbf{X})\|^2$ increases while $\|\text{offdiag}(\mathbf{X})\|^2$ decreases by the same amount.

2.3.2 Symmetrisation

Next, compute and apply a plane rotation with parameters chosen s.t.

$$\begin{bmatrix} c' & s'e^{i\phi'} \\ -s'e^{-i\phi'} & c' \end{bmatrix} \begin{bmatrix} x'_{kk} & x'_{kj} \\ 0 & x'_{jj} \end{bmatrix} = \begin{bmatrix} x''_{kk} & x''_{kj} \\ x''_{jk} & x''_{jj} \end{bmatrix} \tag{15}$$

where $c' = \cos\theta'$, $s' = \sin\theta'$, $x'_{kk}, x'_{jj}, x''_{kk}, x''_{jj} \in \mathbb{R}$ and $x''_{kj} = x''^{*}_{jk}$. This requires

$$c'x'_{kj} + s'e^{i\phi'}x'_{jj} = -s'e^{i\phi'}x'_{kk} \tag{16}$$

We also have

$$x''_{kk} = c'x'_{kk} \tag{17}$$

and

$$x''_{jj} = -s'e^{-i\phi'}x'_{kj} + c'x'_{jj} \tag{18}$$

Clearly x''_{kk} is real. Denoting $x'_{kj} = \left|x'_{kj}\right|e^{i\gamma}$, it can be seen that we require $\phi' = \gamma$ to ensure that x''_{jj} is real. Equation (16) then takes the form

$$c'\left|x'_{kj}\right|e^{i\gamma} + s'e^{i\gamma}x'_{jj} = -s'e^{i\gamma}x'_{kk} \tag{19}$$

which is satisfied by

$$c'\left|x'_{kj}\right| = -s'(x'_{kk} + x'_{jj}) \tag{20}$$

i.e

$$\tan\theta' = -\left|x'_{kj}\right| / (x'_{kk} + x'_{jj}) \tag{21}$$

We denote the embedded rotation of equation (15) by

$$\mathbf{X}'' = \mathbf{Q}(\theta', \phi')\mathbf{X}' \tag{22}$$

where $\mathbf{X}'' \in \mathbb{C}^{m\times n}$ and $\mathbf{Q}(\theta', \phi') \in \mathbb{C}^{m\times m}$. Note once again that $\|\mathbf{X}''\| = \|\mathbf{X}'\|$.

2.3.3 Jacobi transformation

Finally, given the matrix $\mathbf{X}''$ resulting from the symmetrisation, compute and apply a Jacobi transformation with the rotation parameters chosen s.t.

$$\begin{bmatrix} c'' & s''e^{i\phi''} \\ -s''e^{-i\phi''} & c'' \end{bmatrix} \begin{bmatrix} x''_{kk} & x''_{kj} \\ x''_{jk} & x''_{jj} \end{bmatrix} \begin{bmatrix} c'' & -s''e^{i\phi''} \\ s''e^{-i\phi''} & c'' \end{bmatrix} = \begin{bmatrix} x'''_{kk} & 0 \\ 0 & x'''_{jj} \end{bmatrix}$$

where $x''_{kk}, x''_{jj}, x'''_{kk}$ and x'''_{jj} are real and $x''_{kj} = x''^{*}_{jk}$. Since the Hermitian symmetry is preserved, x'''_{kk} and x'''_{jj} are guaranteed to be real. The required rotation parameters are given by

$$\tan 2\theta'' = 2\left|x''_{kj}\right| / (x''_{kk} - x''_{jj}) \tag{23}$$

and $\phi'' = \delta$ where $x''_{kj} = \left|x''_{kj}\right|e^{i\delta}$ [10]. We denote the Jacobi transformation by

$$\mathbf{X}''' = \mathbf{J}'(\theta'', \phi'')\mathbf{X}''\mathbf{J}(\theta'', \phi'') \tag{24}$$

where $\mathbf{J} \in \mathbb{C}^{n\times n}$ is the embedded transformation represented above, $\mathbf{X}''' \in \mathbb{C}^{m\times n}$ and $\mathbf{J}' \in \mathbb{C}^{m\times m}$ takes the form

$$\mathbf{J}' = \begin{bmatrix} \mathbf{J}^H & 0 \\ 0 & \mathbf{I} \end{bmatrix} \tag{25}$$

The complex Kogbetliantz transformation (including the additional triangularisation step) may be expressed in the form

$$\mathbf{X}''' = \mathbf{J}'(\theta'', \phi'')\mathbf{Q}(\theta', \phi')\mathbf{F}(\theta, \phi, \beta)\mathbf{X}\mathbf{J}(\theta'', \phi'') \tag{26}$$

This completes the elementary transformation for the case $j \leq n$. In preparation for the next iteration the input matrix is now updated by setting $\mathbf{X} \leftarrow \mathbf{X}'''$. Note, once again, that $\|\mathbf{X}'''\| = \|\mathbf{X}\|$ and that, as a result of the Kogbetliantz transformation, $\|\text{diag}(\mathbf{X})\|^2$ increases while $\|\text{offdiag}(\mathbf{X})\|^2$ decreases by the same amount. The combined operator $\mathbf{J}'(\theta'', \phi'')\mathbf{Q}(\theta', \phi')\mathbf{F}(\theta, \phi, \beta)$ applied from the left in equation (26) contributes to the overall transformation $\mathbf{U}$ in equation (3) while the operator $\mathbf{J}(\theta'', \phi'')$ applied from the right contributes to $\mathbf{V}$.

The process of locating the dominant off-diagonal element, and then eliminating it using either a Givens rotation (for $j > n$) or a complex Kogbetliantz transformation (for $j \leq n$), is repeated until the dominant off-diagonal element is sufficiently small. Again, as for the conventional Kogbetliantz algorithm [18], this process is guaranteed to converge so that

$$\mathbf{X} \rightarrow \begin{bmatrix} \mathbf{\Sigma} \\ 0 \end{bmatrix} \tag{27}$$

3. GENERALISATION TO POLYNOMIAL MATRICES

Assume, without loss of generality, that we are given a polynomial matrix of the form

$$\underline{\mathbf{X}}(z) = \sum_{t=0}^{T} z^{-t}\mathbf{X}(t) \tag{28}$$

where $\mathbf{X}(t)$ $(t = 0, 1, \ldots T) \in \mathbb{C}^{m\times n}$ and $m > n$. As before, the algorithm for square polynomial matrices may be deduced very simply as a special case of what follows. We seek to compute and apply a transformation of the form

$$\underline{\mathbf{U}}(z)\underline{\mathbf{X}}(z)\underline{\mathbf{V}}(z) = \begin{bmatrix} \underline{\mathbf{D}}(z) \\ 0 \end{bmatrix} \tag{29}$$

where $\underline{\mathbf{D}}(z)$ is an $n \times n$ diagonal polynomial matrix and $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ are paraunitary polynomial matrices of dimension $m \times m$ and $n \times n$ respectively i.e.

$$\underline{\mathbf{U}}(z)\underline{\widetilde{\mathbf{U}}}(z) = \underline{\widetilde{\mathbf{U}}}(z)\underline{\mathbf{U}}(z) = \mathbf{I}_m \tag{30}$$

and

$$\underline{\mathbf{V}}(z)\underline{\widetilde{\mathbf{V}}}(z) = \underline{\widetilde{\mathbf{V}}}(z)\underline{\mathbf{V}}(z) = \mathbf{I}_n \tag{31}$$

The tilde notation denotes paraconjugation of a polynomial matrix as defined in [2].

The algorithm described here generalises the modified Kogbetliantz algorithm, detailed above, to polynomial matrices in a manner similar to that in which the SBR2 algorithm generalises the classical Jacobi algorithm to para-Hermitian polynomial matrices [17]. It begins by transforming the polynomial matrix $\underline{\mathbf{X}}(z)$ so that $\mathbf{X}(0)$, the coefficient matrix of order zero (referred to here as the zero-plane matrix), has real diagonal elements. This is accomplished using a diagonal phase rotation matrix of the type represented by equation (5). The algorithm continues by locating the dominant off-diagonal coefficient in $\underline{\mathbf{X}}(z)$, denoted here by $x_{jk}(t)$, and shifting it to the zero-plane matrix using a pure delay transformation of the form

$$\underline{\mathbf{X}}'(z) = \underline{\mathbf{B}}_1^{(k,t)}(z)\underline{\mathbf{X}}(z)\underline{\mathbf{B}}_2^{(k,t)}(z) \tag{32}$$

where

$$\underline{\mathbf{B}}_1^{(k,t)}(z) = \begin{bmatrix} \mathbf{I}_{k-1} & 0 & 0 \\ 0 & z^{-t} & 0 \\ 0 & 0 & \mathbf{I}_{m-k} \end{bmatrix} \tag{33}$$

and

$$\underline{\mathbf{B}}_2^{(k,t)}(z) = \begin{bmatrix} \mathbf{I}_{k-1} & 0 & 0 \\ 0 & z^{t} & 0 \\ 0 & 0 & \mathbf{I}_{n-k} \end{bmatrix} \tag{34}$$

Note that the diagonal elements are not affected by this transformation which results in $x'_{jk}(0) = x_{jk}(t)$.

Having shifted the dominant off-diagonal coefficient to the zero-plane matrix $\mathbf{X}'(0)$, a modified Kogbetliantz transformation, as defined in section 2, is computed with respect to this matrix in order to drive the dominant coefficient to zero. The sequence of transformations required for the modified Kogbetliantz transformation is then applied to the entire polynomial matrix $\underline{\mathbf{X}}'(z)$, i.e. to every coefficient matrix in $\underline{\mathbf{X}}'(z)$ including $\mathbf{X}'(0)$. In preparation for the next iteration, the transformed polynomial matrix $\underline{\mathbf{X}}'(z)$ is then redesignated as $\underline{\mathbf{X}}(z)$.

The process of locating the dominant off-diagonal coefficient, moving it to the zero-plane, and eliminating it by means of a modified Kogbetliantz transformation, is repeated iteratively until the dominant off-diagonal coefficient is sufficiently small. Noting that $\|\text{diag}(\mathbf{X}(0))\|$ increases monotonically throughout the entire iterative process while $\|\underline{\mathbf{X}}(z)\|$ remains invariant, it can easily be shown that this procedure converges to produce a diagonalised matrix $\underline{\mathbf{\Gamma}}(z)$ of the form specified on the right hand side of equation (29). The proof follows a similar line of argument to that used to prove the convergence proof of the SBR2 algorithm [17].

4. NUMERICAL EXAMPLE

The performance of the PSVD algorithm outlined in section 3 is illustrated here by means of a simple numerical example. A 5×3 polynomial matrix $\underline{\mathbf{X}}(z)$ of order 2 was generated with the real and imaginary components of its complex coefficients selected randomly from a normal distribution with mean 0 and variance 1. This matrix is depicted in figure 1, where the absolute values of the polynomial coefficients are plotted in the 5×3 array of stem plots. Figure 2 provides a similar representation of the diagonalised polynomial matrix $\underline{\mathbf{\Gamma}}(z)$ obtained using the PSVD algorithm after 318 iterations, when the magnitude of the dominant off-diagonal coefficient was < 0.005. The value of this quantity as a function of iteration number is plotted in figure 3 which shows that the convergence of the algorithm is quite rapid but, as expected, non-monotonic [17]. From figure 2 it can be seen that the output matrix $\underline{\mathbf{\Gamma}}(z)$ takes the required form. The value of $\|\text{offdiag}(\underline{\mathbf{\Gamma}}(z))\|^2$ was found to be 0.0005 as compared to the value of $\|\underline{\mathbf{\Gamma}}(z)\|^2$ which was 70.81.

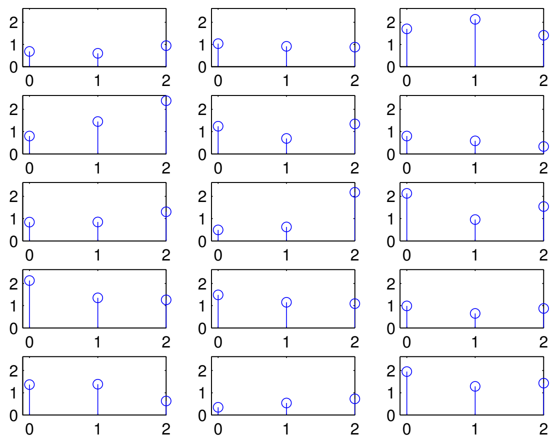
Figure 1: Input 5×3 Polynomial Matrix $\underline{\mathbf{X}}(z)$.

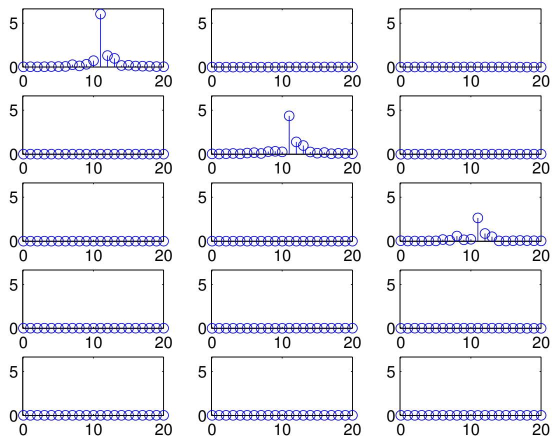
Figure 2: Output matrix $\underline{\mathbf{\Gamma}}(z)$ produced by PSVD algorithm

4.1 Comparison with alternative SBR2 method

An alternative approach to computing the PSVD in equation (29) is to form the polynomial matrix products $\underline{\mathbf{P}}_1(z) = \underline{\mathbf{X}}(z)\underline{\widetilde{\mathbf{X}}}(z)$ and $\underline{\mathbf{P}}_2(z) = \underline{\widetilde{\mathbf{X}}}(z)\underline{\mathbf{X}}(z)$ of dimension 5×5 and 3×3 respectively. Using the SBR2 algorithm to compute the PEVD of $\underline{\mathbf{P}}_1(z)$ and $\underline{\mathbf{P}}_2(z)$ respectively, serves to generate the paraunitary matrices $\underline{\mathbf{U}}(z)$ and $\underline{\mathbf{V}}(z)$ required for the transformation in equation (29). The output matrix $\underline{\mathbf{\Gamma}}(z)$ produced using this alternative method is depicted in figure 4. Once again this can be seen to take the required diagonal form. However, $\|\text{offdiag}(\underline{\mathbf{\Gamma}}(z))\|^2$ now takes the value 2.32 which is much greater than that achieved using the PSVD algorithm of section 3. This illustrates the reduced numerical precision which arises from forming the product matrices in the PEVD method.

5. CONCLUSIONS

An algorithm for computing the PSVD of a complex polynomial matrix, based on a generalisation of the Kogbetliantz

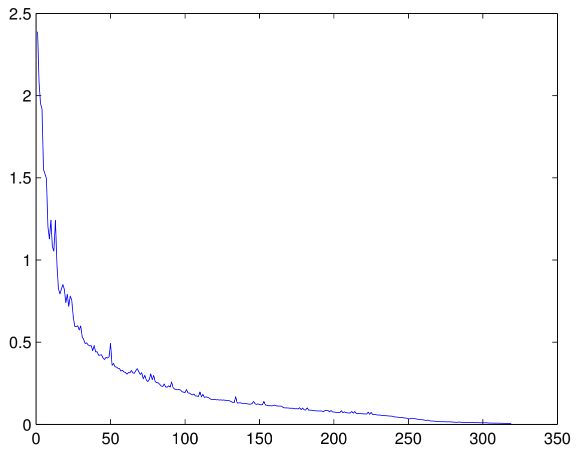
Figure 3: Convergence of PSVD algorithm: magnitude of the dominant off-diagonal coefficient plotted as a function of iteration number.

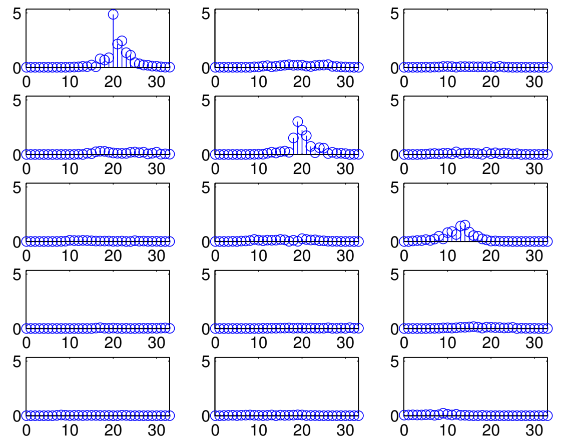
Figure 4: Output matrix $\underline{\mathbf{\Gamma}}(z)$ produced by SBR2 algorithm

method for complex scalar matrices, has been presented. The results obtained for a simple numerical example demonstrate that the new PSVD algorithm can work well in practice. A comparison with the alternative PEVD approach based on the SBR2 algorithm demonstrated, as expected, that the PSVD algorithm generates a more accurate decomposition for the same example.

REFERENCES

[1] T. Kailath. *Linear Systems*. Prentice-Hall International Inc., 1980.

[2] P.P. Vaidyanathan. *Multirate Systems and Filter Banks*. Prentice Hall, 1993.

[3] N. Delfosse and P. Loubaton. Adaptive blind separation of independent sources: A second order stable algorithm for the general case. *IEEE Trans. Circuits and Systems*, 47:1056–1070, July 2000.

[4] R. H. Lambert, M. Joho and H. Mathis. Polynomial singular values for number of wideband sources estimation and principal component analysis. *Proc. Int. Conf. Independent Component Analysis*, pages 379–383, 2001.

[5] S.Y. Kung, Y. Wu and X. Zhang. Bezout space-time precoders and equalisers for MIMO channels. *IEEE Trans. Signal Processing*, 50:2499–2514, Oct 2002.

[6] A. Medles and D.T.M. Slock. Linear precoding for spatial multiplexing MIMO systems: Blind channel estimation aspects. *Proc. IEEE Int Conf on Communications*, pages 401–404, April 2002.

[7] L. Rota, P. Comon and S. Icart. Blind MIMO paraunitary equalizer. *Proc. IEEE Int Conf on Acoustics, Speech and Signal Processing*, April 2003.

[8] P.P. Vaidyanathan. Theory of optimal orthonormal subband coders. *IEEE Trans Signal Processing*, 46(6):1528–1543, 1998.

[9] P. Moulin and M.K. Mihcak. Theory and design of signal-adapted FIR paraunitary filter banks. *IEEE Trans Signal Processing*, 46:920–929, April 1998.

[10] G.H. Golub and C.F. Van Loan. *Matrix Computations (Third Edition)*. The John Hopkins University Press, 1996.

[11] S. Haykin. *Adaptive Filter Theory (Fourth Edition)*. Prentice Hall, 2002.

[12] M. Davies, S. Lambotharan, J.A. Chambers and J.G. McWhirter. Broadband MIMO beamforming for frequency selective channels using the sequential best rotation algorithm. *Proc 67th Vehicular Technology Conf, Singapore*, 2008.

[13] M. Davies, S. Lambotharan, J.A. Foster, J.A. Chambers and J.G. McWhirter. A polynomial QR decomposition based turbo equalization technique for frequency selective MIMO channels. *Proc 69th Vehicular Technology Conf, Barcelona*, 2009.

[14] C.H. Ta and S. Weiss. A design of precoding and equalisation for broadband MIMO transmission. *Proc 15th Int Conf on Digital Signal Processing, Cardiff*, 2007.

[15] S. Weiss, C.H. Ta and C. Liu. A wiener filter approach to the design of filter bank based single-carrier precoding and equalisation. *Power Line Communications and Its Applications*, pages 493–498, 2007.

[16] S. Redif and T. Cooper. Paraunitary filterbank design via a polynomial singular value decomposition. *Proc. IEEE Int Conf on Acoustics, Speech and Signal Processing*, 2005.

[17] J.G. McWhirter, P.D. Baxter, T. Cooper, S. Redif and J.A. Foster. An EVD algorithm for para-Hermitian polynomial matrices. *IEEE Trans Signal Processing*, 55(6):2158–2169, 2007.

[18] G.E. Forsythe and P. Henrici. The cyclic jacobi method for computing the principal values of a complex matrix. *Trans. Amer. Math. Soc.*, 94:1–23, 1960.

[19] J.A. Foster, J.G. McWhirter, M.R. Davies and J.A. Chambers. An algorithm for calculating the QR and singular value decompositions of polynomial matrices. *IEEE Trans Signal Processing*, 58(6), March, 2010.