AN ALGORITHM FOR POLYNOMIAL MATRIX SVD BASED ON GENERALISED KOBETLIANTZ 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: mewhirterjg@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 communications [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 $X \in \mathbb{C}^{m \times n}$, 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 $Q \in \mathbb{C}^{m \times m}$ is computed s.t.

$$QX = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where $R \in \mathbb{C}^{n \times n}$ is an upper triangular matrix with real diagonal elements. Computation of the SVD can then proceed by treating $R$ 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

$$Q_1 RV = \Sigma$$

where $Q_1 \in \mathbb{C}^{n \times n}$ and $V \in \mathbb{C}^{n \times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal. In combination, we have

$$U X V^H = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

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

$$U = \begin{bmatrix} Q_1 & 0 \\ 0 & I \end{bmatrix}$$

and so this constitutes the SVD of $X$ [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 $2 \times 2$ sub-matrix to which it is applied.

© EURASIP, 2010 ISSN 2076-1465
2. MODIFIED KOGBEKTIANZ ALGORITHM FOR COMPLEX NON-SQUARE MATRICES

Assuming that \( m > n \), we seek to compute the SVD of a matrix \( X \in \mathbb{C}^{m \times n} \) 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 \( j \)th column by \( \exp(-i\alpha_j) \) where \( \alpha_j \) 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

\[
X \rightarrow XT
\]

where \( T \in \mathbb{C}^{n \times n} \) is a diagonal matrix of phase rotations and is therefore unitary. Note that \( ||X|| \) 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 \( X \) (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

\[
\begin{bmatrix}
c & -se^{-i\phi} \\-se^{i\phi} & c
\end{bmatrix}
\begin{bmatrix}
x_{kk} \\
x_{jk}
\end{bmatrix}
= \begin{bmatrix}x'_{kk} \\
x'_{jk}
\end{bmatrix}
\]

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

\[
-xse^{-i\phi}x_{kk} + cx_{jk} = 0
\]

while

\[
x'_{kk} = cx_{kk} + se^{i\phi}x_{jk}
\]

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

\[
-xse^{i\omega}x_{kk} + c|x_{jk}|e^{i\omega} = 0
\]

which is satisfied by

\[
\tan\theta = |x_{jk}|/x_{kk}
\]

We denote this transformation by

\[
X' = G(\theta, \phi)X
\]

2.3 Complex Kogbetliantz transformation

If \( j \leq n \) (i.e. if the dominant off-diagonal element of \( X \) lies within the upper \( n \times n \) sub-matrix), perform the following complex Kogbetliantz transformation designed to eliminate both \( x_{jk} \) and \( x_{kj} \). 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

\[
\begin{bmatrix}
c & se^{i\phi} \\se^{-i\phi} & c
\end{bmatrix}
\begin{bmatrix}
x_{kk} \\
x_{kj}
\end{bmatrix}
= \begin{bmatrix}x'_{kk} \\
x'_{kj}
\end{bmatrix}
\]

where \( x_{ij}, x_{kk} \) 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

\[
x'_{jj} = -se^{i\omega}x_{kj} + cx_{jj}
\]

This quantity is not necessarily real, so a further phase adjustment is required. Denote \( x'_{jj} = |x'_{jj}|e^{i\beta} \) and multiply the \( j \)th 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 adjustment and interchange of rows and columns, by

\[
X' = F(\theta, \phi, \beta)X
\]

2.3.2 Symmetrisation

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

\[
\begin{bmatrix}
c' & se^{i\phi'} \\se^{-i\phi'} & c'
\end{bmatrix}
\begin{bmatrix}
x'_{kk} \\
x'_{kj}
\end{bmatrix}
= \begin{bmatrix}x_{kk} \\
x_{kj}
\end{bmatrix}
\]

458
where \( c' = \cos \theta', \ s' = \sin \theta', \ x'_{jk}, x'_{jj}, x''_{kk}, x''_{jj} \subseteq \mathbb{R} \) and \( x''_{jk} = x''_{kj} \). This requires

\[
c' x'_{kj} + s' e^{i \phi'} x'_{jj} = -s' e^{i \phi'} x'_{kk}
\]  

(16)

We also have

\[
x''_{jk} = c' x''_{kk}
\]  

(17)

and

\[
x''_{jj} = -s' e^{-i \phi'} x'_{kj} + c' x'_{jj}
\]  

(18)

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

\[
c' |x'_k| e^{i \eta} + s' e^{i \eta} x'_j = -s' e^{i \eta} x'_{kk}
\]  

(19)

which is satisfied by

\[
c' |x'_k| = -s' (x'_{kk} + x'_j)
\]  

(20)

i.e.

\[
\tan \theta' = - |x'_k| / (x'_{kk} + x'_j)
\]  

(21)

We denote the embedded rotation of equation (15) by

\[
X'' = Q(\theta', \phi') X'
\]  

(22)

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

2.3.3 Jacobi transformation

Finally, given the matrix \( X'' \) 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 \( x''_{kk}, x''_{kj}, x''_{jj} \) and \( x''_{kj} \) 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

\[
\tan 2 \theta'' = 2 |x''_{kj}| / (x''_{kk} + x''_{jj})
\]  

(23)

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

\[
X''' = J'(\theta'', \phi'') X'' J(\theta'', \phi'')
\]  

(24)

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

\[
J' = \begin{bmatrix}
    J'' & 0 \\
    0 & I
\end{bmatrix}
\]  

(25)

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

\[
X''' = J'(\theta'', \phi'') Q(\theta', \phi') F(\theta, \phi, \beta) X J(\theta'', \phi'')
\]  

(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 \( X \leftarrow X''' \). Note, once again, that \( \|X''\| = \|X\| \) and that, as a result of the Kogbetliantz transformation, \( \|\text{diag}(X)\|^2 \) increases while \( \|\text{offdiag}(X)\|^2 \) decreases by the same amount. The combined operator \( J'(\theta'', \phi'') Q(\theta', \phi') F(\theta, \phi, \beta) \) applied from the left in equation (26) contributes to the overall transformation \( U \) in equation (3) while the operator \( J(\theta'', \phi'') \) applied from the right contributes to \( V \).

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

\[
X \rightarrow \begin{bmatrix}
    \Sigma \\
    0
\end{bmatrix}
\]  

(27)

3. GENERALISATION TO POLYNOMIAL MATRICES

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

\[
X(z) = \sum_{t=0}^{T} z^{-t} X(t)
\]  

(28)

where \( X(t) \) \( (t = 0, 1, \ldots T) \subseteq \mathbb{C}^{m \times n} \) 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

\[
U(z) X(z) V(z) = \begin{bmatrix}
    D(z) \\
    0
\end{bmatrix}
\]  

(29)

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

\[
U(z) U(z) = V(z) V(z) = I_m
\]  

(30)

and

\[
V(z) V(z) = \overline{V(z)} V(z) = I_n
\]  

(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 \( X(z) \) so that \( X(0) \), 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 \( X(z) \), denoted here by \( x_{jk}(t) \),
and shifting it to the zero-plane matrix using a pure delay transformation of the form
\[
X'(z) = B_1^{(k_j)}(z) X(z) B_2^{(k_j)}(z)
\]  
(32)

where
\[
B_1^{(k_j)}(z) = \begin{bmatrix}
    I_{k-1} & 0 & 0 \\
    0 & z^{-t} & 0 \\
    0 & 0 & I_{m-k}
\end{bmatrix}
\]  
(33)

and
\[
B_2^{(k_j)}(z) = \begin{bmatrix}
    I_{k-1} & 0 & 0 \\
    0 & z' & 0 \\
    0 & 0 & I_{n-k}
\end{bmatrix}
\]  
(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 \(X'(0)\), 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 \(X'(z)\), i.e. to every coefficient matrix in \(X'(z)\) including \(X'(0)\). In preparation for the next iteration, the transformed polynomial matrix \(X'(z)\) is then redesigned as \(X(z)\).

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 \(\| \text{diag}(X(0)) \|\) increases monotonically throughout the entire iterative process while \(\| X(z) \|\) remains invariant, it can easily be shown that this procedure converges to produce a diagonalised matrix \(\Sigma(z)\) 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 \(5 \times 3\) polynomial matrix \(\bar{X}(z)\) 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 \(5 \times 3\) array of stem plots. Figure 2 provides a similar representation of the diagonalised polynomial matrix \(\Sigma(z)\) 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 \(\Sigma(z)\) takes the required form. The value of \(\| \text{offdiag}(\Sigma(z)) \|^2\) was found to be 0.0005 as compared to the value of \(\| \Sigma(z) \|^2\) which was 70.81.

4.1 Comparison with alternative SBR2 method

An alternative approach to computing the PSVD in equation (29) is to form the polynomial matrix products \(P_1(z) = \bar{X}(z) \bar{X}(z)\) and \(P_2(z) = \bar{X}(z) \bar{X}(z)\) of dimension \(5 \times 5\) and \(3 \times 3\) respectively. Using the SBR2 algorithm to compute the PEVD of \(P_1(z)\) and \(P_2(z)\) respectively, serves to generate the paraunitary matrices \(U(z)\) and \(V(z)\) required for the transformation in equation (29). The output matrix \(\Sigma(z)\) produced using this alternative method is depicted in figure 4. Once again this can be seen to take the required diagonal form. However, \(\| \text{offdiag}(\Sigma(z)) \|^2\) 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 $\mathbf{Z}(z)$ 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