AN ALGORITHM FOR COMPUTING THE QR DECOMPOSITION OF A POLYNOMIAL MATRIX

Joanne Foster [1], John McWhirter [1,2] and Jonathon Chambers [1]

[1] Cardiff University, [2] QinetiQ, Malvern

ABSTRACT

This paper introduces an algorithm for computing a QR decomposition of a polynomial matrix. The algorithm proceeds to perform the decomposition by following the same strategy in eliminating entries of the matrix as is used in the Givens method for a QR decomposition of a scalar matrix. However scalar Givens rotation matrices can no longer be applied. Instead, a polynomial Givens rotation is introduced, enabling the QR decomposition of a polynomial matrix. Convergence of the algorithm is discussed and through simulations the capability of the algorithm is assessed.

Index Terms— Polynomial matrix, paraunitary matrix, polynomial matrix QR decomposition

1. INTRODUCTION

Polynomial matrices have many applications in the field of control, but in recent years they have also been used extensively in the areas of digital signal processing and communications. Examples of their applications include broadband adaptive sensor array processing, MIMO communication channels, broadband subspace decomposition and also digital filter banks for subband coding or data compression [1,2]. In the context of this paper, polynomial matrices arise 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.

If, instead, the received signals are instantaneously mixed, then there are no time delays in the propagation of the signals from the sources to the sensors and a scalar matrix is sufficient to describe the mixing. In this situation the QR decomposition algorithm can be used for identifying communication channels based on second order statistics. This would involve working directly on the data matrix, with no need to form a covariance matrix for the signals, as required if using the Eigenvalue Decomposition (EVD). This technique can be extended to broadband signal processing, where polynomial matrices are now observed, by using a suitable algorithm for computing the QR factorisation of a polynomial matrix.

This paper firstly discusses polynomial matrices and the QR factorisation of a scalar matrix. Subsequently the concept of a polynomial Givens Rotation is introduced, before detailing the Polynomial QR algorithm. Finally through simulations the capability of the algorithm is demonstrated.

2. 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 polynomial matrix $\underline{\mathbf{A}}(z)$, where the indeterminate variable of the polynomial is, in this case, z^{-1} used to represent a unit delay, can be expressed as follows: -

$$\underline{\mathbf{A}}(z)=\begin{bmatrix} \underline{a}_{11}(z) & \cdots & \underline{a}_{1q}(z) \\ \vdots & & \vdots \\ \underline{a}_{p1}(z) & \cdots & \underline{a}_{pq}(z) \end{bmatrix}=\sum_{t=t_1}^{t_2}\mathbf{A}(t)z^{-t} \qquad (1)$$

where $t\in Z, t_1\le t_2$ and $\mathbf{A}(t)\in C^{p\times q}$ is the matrix of coefficients to z^{-t}. The polynomial coefficient in the $(i,j)^{\text{th}}$ element of $\underline{\mathbf{A}}(z)$ corresponding to a delay of z^{-t} will be denoted as $a_{ij}(t)$ and the order of the polynomial matrix can be calculated as $(t_2 - t_1)$, where the values of the parameters t_1 and t_2 are not necessarily positive. The underline notation in equation 1 is used to denote a polynomial, whether it is a matrix, vector or scalar, to avoid confusion with the notation used for the z-transform of a variable. Let the set of polynomial matrices, with complex coefficients, be denoted by $\underline{C}^{a\times b}$ where a denotes the number of rows and b the number of columns.

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

$$\underline{\tilde{\mathbf{A}}}(z)=\underline{\mathbf{A}}_*^{\mathrm{T}}(1/z) \qquad (2)$$

where * denotes the complex conjugation of the coefficients of each polynomial element and ^{T} denotes matrix transposition. A polynomial matrix $\underline{\mathbf{A}}(z)$ is said to be paraunitary if the following is true:

$$\underline{\mathbf{A}}(z)\widetilde{\underline{\mathbf{A}}}(z) = \widetilde{\underline{\mathbf{A}}}(z)\underline{\mathbf{A}}(z) = \mathbf{I} \tag{3}$$

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

$$\left\|\underline{\mathbf{A}}(z)\right\|_F = \sqrt{\sum_{t=t_1}^{t_2}\sum_{i=1}^{p}\sum_{j=1}^{q}\left|a_{ij}(t)\right|^2}\ . \tag{4}$$

3. QR FACTORISATION OF A SCALAR MATRIX

The QR factorization of a scalar matrix $\mathbf{A} \in C^{p\times q}$ is given as follows:

$$\mathbf{A} = \mathbf{QR} \tag{5}$$

where $\mathbf{Q} \in C^{p\times p}$ is a unitary matrix and $\mathbf{R} \in C^{p\times q}$ is an upper triangular matrix. One method for computing the unitary matrix $\mathbf{Q}$ is by calculating a series of plane rotations, where each rotation will drive one of the elements beneath the diagonal to zero, [3]. The elements of the matrix below the diagonal are rotated to equal zero in a particular order to ensure that each element need only be eliminated once. There are several different orderings that can be implemented. However, for the purposes of this paper the elements are eliminated starting with the uppermost left element and then moving across elements beneath the diagonal in each row from left to right, before moving to the next row down.

This technique of eliminating the elements beneath the diagonal in a specified order can be directly applied to polynomial matrices, even though each element now consists of a series of polynomial coefficients. All coefficients from the series need to be eliminated to ensure that the polynomial element is zero. Unfortunately this can no longer be achieved with a scalar Givens rotation matrix; instead a polynomial Givens rotation is required.

4. A POLYNOMIAL GIVENS ROTATION

An elementary polynomial Givens rotation (EPGR) takes the form of a Givens rotation preceded by an elementary time shift matrix as follows:

$$\begin{aligned}\hat{\underline{\mathbf{G}}}^{(\alpha,\theta,\phi,t)}(z) &= \begin{bmatrix} ce^{i\alpha} & se^{i\phi} \\ -se^{-i\phi} & ce^{-i\alpha} \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & z^t \end{bmatrix} \\ &= \begin{bmatrix} ce^{i\alpha} & se^{i\phi}z^t \\ -se^{-i\phi} & ce^{-i\alpha}z^t \end{bmatrix}\end{aligned} \tag{6}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$. The aim of this matrix, when applied to the polynomial vector $\underline{a}(z) \in \underline{C}^{2\times 1}$ as demonstrated

$$\begin{bmatrix} ce^{i\alpha} & se^{i\phi}z^t \\ -se^{-i\phi} & ce^{-i\alpha}z^t \end{bmatrix}\begin{bmatrix} \underline{a}_1(z) \\ \underline{a}_2(z) \end{bmatrix} = \begin{bmatrix} \underline{a}'_1(z) \\ \underline{a}'_2(z) \end{bmatrix} \tag{7}$$

is to drive the polynomial coefficient $a_2(t)$ to zero so that $a'_2(0) = 0$. This condition is satisfied when the rotation angles are chosen as follows:

$$\tan\theta = \frac{\left|a_2(t)\right|}{\left|a_1(0)\right|},\ \phi = -\arg(a_2(t)) \text{ and } \alpha = -\arg(a_1(0)). \tag{8}$$

Furthermore, following the application of the EPGR $a'_1(0)$ is real and $\left|a'_1(0)\right|^2 = \left|a_1(0)\right|^2 + \left|a_2(t)\right|^2$.

A series of EPGR's can be applied iteratively to the vector $\underline{a}(z)$, as demonstrated in equation 7, to drive all coefficients of the polynomial element $\underline{a}_2(z)$ arbitrarily close to zero. At each iteration the rotation angles θ, ϕ and α are chosen to zero the coefficient within $\underline{a}_2(z)$ with maximal 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 element may be chosen. The complete series of EPGR's required constitutes a complete polynomial Givens rotation, which will be denoted by the matrix $\underline{\mathbf{G}}(z)$ with:

$$\underline{\mathbf{G}}(z)\begin{bmatrix} \underline{a}_1(z) \\ \underline{a}_2(z) \end{bmatrix} \cong \begin{bmatrix} \underline{a}'_1(z) \\ 0 \end{bmatrix} \tag{9}$$

where all coefficients of the polynomial element $\underline{a}_2(z)$ have been driven arbitrarily close to zero over a series of EPGR's. The polynomial Givens rotation matrix can therefore be calculated as follows:

$$\underline{\mathbf{G}}(z) = \hat{\underline{\mathbf{G}}}_i(z).....\hat{\underline{\mathbf{G}}}_1(z) \tag{10}$$

where i is the number of EPGR's required to drive all coefficients of the polynomial element sufficiently close to zero. Note that each of the EPGR's is paraunitary and so the complete polynomial Givens rotation will also be paraunitary.

In practice, it is not feasible to drive all coefficients of a polynomial element to zero, instead the coefficients are reduced until the magnitude of the dominant coefficient in $\underline{a}_2(z)$ is sufficiently small, i.e.

$$\left|a_2(t)\right| < \varepsilon \tag{11}$$

where $\varepsilon > 0$ is a pre-specified small value. A proof of convergence for a complete polynomial Givens rotation step can now be easily deduced.

With every application of an EPGR, to zero the dominant coefficient $a_2(t)$, the quantity $\left|a'_1(0)\right|^2$ will increase by the magnitude squared of the largest coefficient within the element. Furthermore, this quantity is bounded above by

the squared Frobenius norm of the vector $\underline{a}(z)$, $\|\underline{a}(z)\|_F^2$, which remains constant throughout all iterations of the algorithm. As $|a_1'(0)|^2$ is monotonically increasing and bounded above, over a series of EPGR's the condition set by equation 11 is guaranteed and the complete polynomial Givens rotation converges in this respect.

A complete polynomial Givens rotation, as shown in equations 9 and 10, can similarly be applied to a polynomial matrix to drive one of the polynomial elements beneath the diagonal to zero. This technique forms the basis of the algorithm for performing the QR decomposition of a polynomial matrix.

5. THE QR DECOMPOSITION OF A POLYNOMIAL MATRIX

The Polynomial QR algorithm is a technique for factorising a polynomial matrix into an upper triangular and a paraunitary polynomial matrix. Let $\underline{\mathbf{A}}(z) \in \underline{C}^{p \times q}$ then the objective of the Polynomial QR algorithm is to find a paraunitary matrix $\underline{\mathbf{Q}}(z) \in \underline{C}^{p \times p}$ such that:

$$\underline{\mathbf{Q}}(z)\underline{\mathbf{A}}(z) = \underline{\mathbf{R}}(z) \tag{12}$$

where $\underline{\mathbf{R}}(z) \in \underline{C}^{p \times q}$ is an upper triangular polynomial matrix. The polynomial matrix $\underline{\mathbf{Q}}(z)$ is computed as a series of complete polynomial Givens rotations, each one designed to drive all coefficients of one of the polynomial elements situated beneath the diagonal to be sufficiently small. The order in which the polynomial elements are eliminated is the same as that of the scalar Givens QR method explained in section 3 of this paper. The algorithm can therefore be thought of as an extension of the conventional Givens QR method for factorising a scalar matrix by applying a series of unitary rotation matrices.

The polynomial QR algorithm operates in a finite number of steps, where at each step one complete polynomial Givens rotation matrix is applied to the polynomial matrix $\underline{\mathbf{A}}(z)$ as follows:

$$\underline{\mathbf{A}}'(z) = \underline{\mathbf{G}}^{(j,k)}(z)\underline{\mathbf{A}}(z) \tag{13}$$

where the indices j and k define the position of the polynomial element that the complete Givens rotation is attempting to eliminate.

Each step, however, consists of an iterative process, where at each iteration an EPGR is calculated, designed to drive the dominant coefficient of the particular polynomial element, $\underline{a}_{jk}(z)$, to zero. This is achieved by rotating $\underline{a}_{jk}(z)$ with the corresponding diagonal element $\underline{a}_{kk}(z)$. At each iteration the EPGR takes the form of a pxp 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 2x2 sub-matrix $\hat{\underline{\mathbf{G}}}^{(\alpha,\theta,\phi,t)}(z)$ shown in equation 6, where the rotation angles θ, ϕ and α are chosen, according to equation 8, to zero the dominant coefficient. Unlike the number of steps, each step does not take a fixed number of iterations. Instead each step runs until all coefficients from the element are sufficiently small and so the stopping condition demonstrated by equation 11 is satisfied. Once all coefficients of the particular polynomial element are sufficiently small, the algorithm moves onto the next polynomial element in the ordering replacing $\underline{\mathbf{A}}(z)$ with $\underline{\mathbf{A}}'(z)$.

The QR algorithm for scalar matrices drives all elements below the diagonal to zero. However the Polynomial QR algorithm, although driving the dominant coefficient at each iteration to zero, only ensures that all coefficients of an element are suitably small before moving on 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, forcing them to increase in magnitude. For this reason a proof of convergence for the algorithm does not simply follow from the proof of convergence given for one polynomial element. A complete proof has still to be found. However, no numerical problems have been encountered when applying the algorithm to a wide range of matrices.

7. RESULTS

A polynomial matrix $\underline{\mathbf{A}}(z) \in \underline{C}^{4 \times 3}$ of order 4 was generated and then used as input to the polynomial QR algorithm. The complex and real parts of the coefficients of each of the elements were randomly drawn from a normal distribution with mean zero and unit variance. Graphical representations for the polynomial output matrices to the algorithm, $\underline{\mathbf{R}}(z)$ and $\underline{\mathbf{Q}}(z)$, can be seen in figures 1 and 2 respectively, where a stem plot is used to demonstrate the amplitude of the series of coefficients for each of the polynomial elements. The value of ε, in equation 11, used to obtain these results was 0.01, allowing the algorithm to run for a total of 124 iterations over all 6 steps.

With each delay step, at each iteration of the algorithm, the order of the polynomial matrices $\underline{\mathbf{Q}}(z)$ and $\underline{\mathbf{R}}(z)$ increases, often after a series of iterations becoming unnecessarily large. Therefore throughout the algorithm the polynomial matrices were truncated, to stop their orders from becoming unnecessarily large and the algorithm slow to implement. The method of truncation is similar to that used in [1] and [4]. A more detailed explanation to why the order can become unnecessarily large is also found in these papers. A suitable truncation method for a polynomial matrix $\underline{\mathbf{A}}(z)$, with 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 the following condition is satisfied:

$$\frac{\sum_{\tau=t_1}^{T_1}\sum_{l=1}^{p}\sum_{m=1}^{q}\left|a_{lm}(\tau)\right|^2}{\left\|\underline{\mathbf{A}}(z)\right\|_F^2} \le \frac{c}{2} \text{ and } \frac{\sum_{\tau=T_2}^{t_2}\sum_{l=1}^{p}\sum_{m=1}^{q}\left|a_{lm}(\tau)\right|^2}{\left\|\underline{\mathbf{A}}(z)\right\|_F^2} \le \frac{c}{2} \quad (14)$$

then the coefficient matrices $\mathbf{A}(\tau)$ for $\tau = t_1,\ldots, T_1$ and $T_2,\ldots, t_2$ can be trimmed from the matrix. For the results demonstrated in this paper, a value of $c = 0.0001$ was used. This ensured that the order of the matrices were no longer unnecessarily large, but also that a good level of matrix decomposition was achieved. The relative error for the decomposition is defined to be:

$$\frac{\left\|\underline{\mathbf{A}}(z) - \underline{\widetilde{\mathbf{Q}}}(z)\underline{\mathbf{R}}(z)\right\|_F}{\left\|\underline{\mathbf{A}}(z)\right\|_F}. \quad (15)$$

This measure was calculated for the results demonstrated and was found to be 0.033.

The Frobenius norm, L, of all polynomial elements beneath the diagonal of the matrix $\underline{\mathbf{A}}(z)$ is plotted over the series of 124 iterations and is shown in figure 3. The reduction of this measure, L, at each of the 6 steps, to zero one of the polynomial elements beneath the diagonal, can be seen in this graph. Clearly, by the final iteration of the algorithm, the matrix is suitably upper triangular with L = 0.19.

8. CONCLUSION

A QR algorithm suitable for polynomial matrices has been introduced. In a similar approach to the Givens QR method for scalar matrices, the algorithm operates in a fixed number of steps. However, unlike the scalar method, each step in the polynomial QR algorithm must now operate as an iterative process. A proof of convergence for each step has been demonstrated in this paper.

Other QR algorithms suitable for polynomial matrices have been developed; work is still being carried out on the convergence of these algorithms and examining the uniqueness of the solutions.

9. REFERENCES

[1] J.G. McWhirter, P.D. Baxter, T. Cooper, S. Redif and J. Foster, "An EVD Algorithm for Para-Hermitian Polynomial Matrices", *IEEE Transactions on Signal Processing*, vol. 55, No. 6, May 2007.

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

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

[4] 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.*

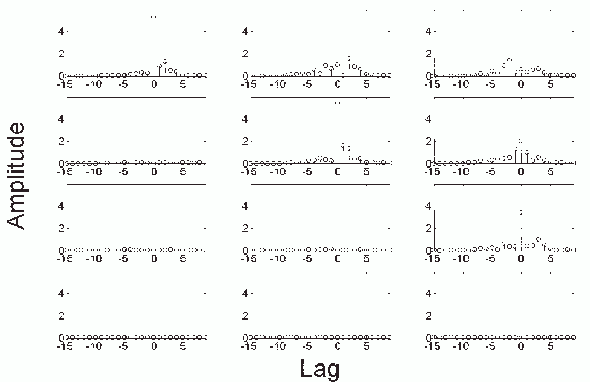

Figure 1 – The upper triangular matrix $\underline{\mathbf{R}}(z)$ obtained from the algorithm after 124 iterations.

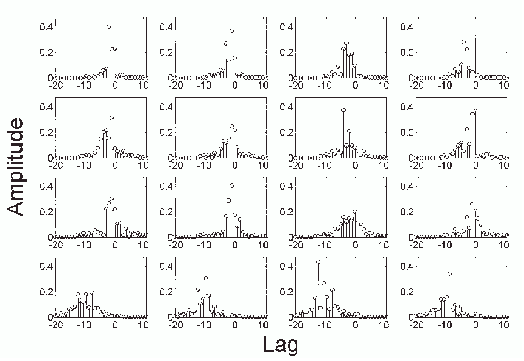

Figure 2 – The paraunitary matrix $\underline{\mathbf{Q}}(z)$ obtained from the algorithm.

Figure 3 – The Frobenius norm of all of the polynomial elements beneath the diagonal, L, plotted over the series of iterations.