A Fast Algorithm for Signal Subspace Tracking, Based on the Jacobi Method.

Bruno Cosenza de Carvalho[1] and Jacques Szczupak[2]
[1,2]Departamento de Engenharia Elétrica - Pontifícia Universidade Católica do Rio de Janeiro, RJ, Brazil
[1]Instituto de Pesquisa e Desenvolvimento, Rio de Janeiro, RJ, Brazil
[2]Engenho Pesquisa, Desenvolvimento e Consultoria, RJ, Brazil
[1]cbruno@ipd.eb.mil.br, [2]jsz@uol.com.br
[1]tel: 55-21-24106271, [1]fax: 55-21-24106270

Abstract–**This work presents a new fast computational algorithm for decomposing a square signal symmetrical matrix, based on the Jacobi method. It is applied to real time systems with data acquisition updated almost sample by sample. The fast computational technique first involves the Data Matrix formation stage, so as to obtain the desired updated data matrix symmetry. In a second step it is developed the final fast algorithm implementation. A comparison is presented with the Golub-Kahan step algorithm, indicating gains and limitations of the proposed approach.**

I. INTRODUCTION

The increasing use of "subspace signal processing" [1] is already a reality. Among the many factors that contribute to this phenomenon is the widely use of orthogonal codes in digital spread spectrum techniques modulation processes besides sensor array processing and many other estimation problems. This type of analysis is specially indicated to noise and interference environments, where it achieves excellent result [2]. On the other hand, since the procedures are usually based on the SVD (Singular Value Decomposition) algorithm [3,4], it is computationally involved; severely limiting it's on line application.

Recently published papers dealt with this type of problem. Eldén and Sjöström [5] focused the problem from the Toeplitz matrices point of view. This situation is quite common when autocorrelation matrices of acquired data are focused. Pango and Champagne [6] presented a method applying Givens rotation to matrices of increasing dimensions with time, having the form:

$$\mathbf{A}(n)=\begin{bmatrix}\sqrt{\lambda}\mathbf{A}(n-1)\\ \mathbf{x}^{H}(n)\end{bmatrix}, \qquad (1)$$

where $\mathbf{x}^{H}(n)$ is the input vector at time n and λ is a real positive number less than 1. Vanpoucke and Moonen [7] use this same matrix model (Eq. (1)) directly applied to the data matrix. Stewart [8] also worked with the updating Data Matrix, described in Eq. (1), applying the URV decomposition. Moonen, Van Dooren and Vanderwale [9] use the QR-updating scheme. Therefore, afterwards, it needs a SVD step. Basically all of the previous methods don't work with limiting the shape of the Data Matrix in order to accelerate the computation process.

This paper proposes a new and faster matrix decomposition procedure, based on Jacobi method [1]. Looking for real time solutions, this approach deals with the data acquisition itself, avoiding the autocorrelation matrix step. Actually the Data Matrix is considered as initialized by the first five data samples, increasing size with subsequent samples, up to a limiting $M\times M$ dimension. The proposed approach kind of adapts from the previous decomposition stage to an updated version for every pair of new samples.

II. THE DATA ACQUISITION MATRIX MODEL

The main objective of this work is to get a new Data Matrix decomposition algorithm recursively, using the previous decomposition to build a conveniently close estimate yielding a fast update. Actually, the proposed approach updates the Data Matrix decomposition for every new pair of data samples to be included in the Data Matrix.

The approach is based on the Jacobi Algorithm, but in order to achieve the desired results it is necessary to work with symmetrical data matrices at every stage of the process. Therefore, it is necessary to describe the data samples in the form of a symmetric matrix, maintaining symmetry with updating new data samples. Due to the symmetry, the Data Matrix initiates composed by the first five samples, increasing size with subsequent samples, up to a limit dimension, *M*. From this point on the algorithm incorporates new samples, preserving matrix symmetry and size.

The matrix updating process is performed for every pair of new samples. Therefore the matrix is updated for n=1,3,5,7...The $k\times k$ dimention input Data Matrix $\mathbf{A}_k$, will have the form

$$\mathbf{A}_k(n)=\begin{bmatrix}x(n-2k+2) & \cdots & x(n-k+1)\\ \vdots & \ddots & \vdots\\ x(n-k+1) & \cdots & x(n)\end{bmatrix} \qquad (2)$$

According to Eq. (2), it is assumed the first sampled Data Matrix is size 3×3 $(k=3)$, containing the first initial five samples. New samples are incorporated according to Eq. (2), such that for

$$\mathbf{A}_{k+1}(n)=\begin{bmatrix}\mathbf{A}_k(n-2) & x(n-k)\\ & \vdots\\ x(n-k)\ \cdots & x(n)\end{bmatrix} \tag{3}$$

When reaching a desired size ($M\times M$) the matrix will maintain its dimension. As such, for every last line and column inclusion, the first line and column are eliminated. This is possible for every pair of new samples. Therefore,

$$\mathbf{A}_M(n)=\begin{bmatrix}\mathbf{A}_{M-1}(n-2) & x(n-M+1)\\ & \vdots\\ x(n-M+1)\ \cdots & x(n)\end{bmatrix} \tag{4}$$

For

$$\mathbf{x}_L=\begin{bmatrix}x(n-k) & \cdots & x(n-1)\end{bmatrix}^T \tag{5}$$

Equation (3) may be written as:

$$\mathbf{A}_{k+1}(n)=\begin{bmatrix}\mathbf{A}_k(n-2) & \mathbf{x}_L\\ \mathbf{x}_L^T & x(n)\end{bmatrix} \tag{6}$$

Let $\mathbf{V}_0$ be the matrix eigenvector for the Schur[1] decomposition of $\mathbf{A}_k(n-2)$, therefore

$$\mathbf{D}_0=\mathbf{V}_0\mathbf{A}_k(n-2)\mathbf{V}_0^T, \tag{7}$$

where $\mathbf{D}_0$ is a diagonal matrix. Defining $\mathbf{V}_1$ as

$$\mathbf{V}_1=\begin{bmatrix}\mathbf{V}_0 & \vdots & \mathbf{0}\\ \cdots & \vdots & \cdots\\ \mathbf{0} & \vdots & 1\end{bmatrix} \tag{8}$$

and applying it to $\mathbf{A}_{k+1}(n)$, it follows from Eq. (4) that

$$\mathbf{D}_1^q=\begin{bmatrix}\mathbf{V}_0 & \vdots & \mathbf{0}\\ \cdots & \vdots & \cdots\\ \mathbf{0} & \vdots & 1\end{bmatrix}\begin{bmatrix}\mathbf{A}_k(n-2) & \mathbf{x}_L\\ \mathbf{x}_L^T & x(n)\end{bmatrix}\begin{bmatrix}\mathbf{V}_0^T & \vdots & \mathbf{0}\\ \cdots & \vdots & \cdots\\ \mathbf{0} & \vdots & 1\end{bmatrix} \tag{9a}$$

$$\mathbf{D}_1^q=\begin{bmatrix}\mathbf{D}_0 & \vdots & \mathbf{V}_0\mathbf{x}_L\\ \cdots & \vdots & \cdots\\ \mathbf{x}_L^T\mathbf{V}_0^T & \vdots & x(n)\end{bmatrix} \tag{9b}$$

Therefore, the procedure yields $\mathbf{D}_1^q$ a quasi-diagonal matrix, except for the added last line and column.

However, although not diagonal, the matrix is still symmetric. The Jacobi method is specially tailored for decomposing this type of square, symmetrical matrix, with a large number of off-diagonal zeros.

Equation (8) indicates a good estimate for the $\mathbf{A}_{k+1}(n)$ eigenvectors. This fact is responsible for saving a large number of iteration stages in subsequent evaluations.

When the data acquisition matrix has already achieved the limiting size *M*, the considered estimate of the new eigenvector matrix is simply the old eigenvector matrix. This type of choice has shown to present positive effects, improving the evaluation efficiency.

III. The Jacobi Method

The Jacobi method for the evaluation of eigenvalues and eigenvectors is particularly interesting for symmetrical square matrices. It accomplishes updating of the type $\mathbf{A}\leftarrow\mathbf{J}^T\mathbf{A}\mathbf{J}$ in an iterative way, such that, the new matrix **A** has a form closer to a diagonal matrix than its predecessor does. Another relevant characteristic of the Jacobi algorithm is the high degree of parallelism in its implementation. Therefore, if processing speed is the final target, one may add to the results obtained in this study the use of a parallel processor structure.

The Jacobi Method continuously applies rotations to the matrix to be decomposed. The rotation matrix is identical to the Givens rotation [3], having the form

$$\mathbf{J}(i,j,\theta)=\begin{bmatrix}1 & \cdots & 0 & \cdots & 0 & \cdots & 0\\ \vdots & \ddots & \vdots & & \vdots & & \vdots\\ 0 & \cdots & c & \cdots & s & \cdots & 0\\ \vdots & & \vdots & \ddots & \vdots & & \vdots\\ 0 & \cdots & -s & \cdots & c & \cdots & 0\\ \vdots & & \vdots & & \vdots & \ddots & \vdots\\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1\end{bmatrix}\begin{matrix}\\ \\ i\\ \\ j\\ \\ \\ \end{matrix} \tag{10}$$

$$\qquad\qquad i \qquad\quad j$$

This way, given a pair (i,j), where $1\le i\le j\le M$, it is enough to find the pair sine/cosine (s,c) that diagonalises the matrix **A** by

$$\begin{bmatrix} b_{ii} & 0 \\ 0 & b_{jj} \end{bmatrix} \leftarrow \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \quad (11)$$

The algorithm for estimating the pair s-c can be easily found in the literature [3].

Another advantage presented by Jacobi's approach, exploring the preserved matrix symmetry, is that the updating of the matrix **A** can be performed in 6k operations (where k is the dimension of this square matrix). It is also common to establish a criterion to stop the Jacobi algorithm. This is made based on largest off-diagonal entry of matrix **A** defined as

$$off(\mathbf{A}) = \sqrt{\sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} a_{ij}^2}, \quad (12)$$

the "norm" of the off-diagonal elements of $k \times k$ matrix **A**, where a_{ij} represents the matrix elements. The tests, using Eq. (12), should be made after completing N Jacobi matrix updates, where N is given by

$$N = k(k-1)/2 \quad (13)$$

to guarantee quadratic convergence in the process [3].

IV. The Proposed Algorithm

The Jacobi Algorithm, used in this study, was of the Cyclic-by-Row type. In this case there is a sequence of previously chosen pairs (i,j) that decreases the computational cost for decomposition updates. The number of floating point operations required by this algorithm is of O(k) flops. For instance, the proposed sequence [3] for the case k=4 is:

$$(i, j) = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4),(1,2),\ldots\}$$

The algorithm has two phases: first the data acquisition matrix will increase in size up to $M \times M$. During the second phase the Data Matrix has a fixed size $M \times M$. The difference between these two phases is how to work with the old eigenvector matrix during the update. This is detailed in the following steps:

Phase 1

1. Acquire the first 5 samples to set up the 3×3 Data Matrix, according to Eq. (1)
2. Evaluate the corresponding matrix eigenvalues and eigenvectors using the Cyclic by Row Jacobi Algorithm for symmetrical square matices.
3. Acquire two new data samples, updating the Data Matrix to the 5×5 dimension
4. Update eigenvalues and eigenvectors starting from estimates given by Eq. (9), using the previously evaluated eigenvectors and eigenvalues. This is done updating the old eigenvector matrix according to Eq. (7).
5. Repeat items 4 and 5, sequentially, until the final Data Matrix dimension is *M* (previously determined value)

Phase 2

6. Discard the two oldest data samples by deleting the first line and column of the old Data Matrix. The matrix dimension M is recovered by bordering this reduced dimension matrix, M-1, with a new last line and column according to Eq. (2)., incorporating the two new data samples.
7. Evaluate the new data acquisition matrix eigenvectors and eigenvalues using the Cyclic by Row Jacobi Algorithm for symmetrical square matrices, starting from the eigenvectors previously evaluated as indicated in Eqs. (9).

V. Illustrative Case

In this section some comparative results are presented, using the number of floating point operations (flops) as a comparison basis. The proposed technique is compared to an equivalent implementation based on Golup-Kahan step approach [3,4]. A number of cases were considered, always using the linear combination of six different sinusoids as the clean input sequence. The final input sequence results from contamination by additive white Gaussian noise, yielding a final 15 dB signal to noise ratio.

Figures 1 to 4 represent the comparative performance, respectively for *M*=10, 20, 30 and 40, as a function of data samples. For all considered cases the total mean square error evaluated on the difference between final eigenvalues computed by both methods, proposed and Golub-Kahan step, was less than 10^{-7}.

In all cases the proposed approach presented computational gains with respect to the Golub-Kahan technique. The gain is more accentuated for *M*=10, reducing with increased matrix dimensions.

VI. Conclusions

This paper proposes a new approach to the eigenvalue-eigenvector evaluation of data structured small and medium dimension symmetric matrices for real time processing.

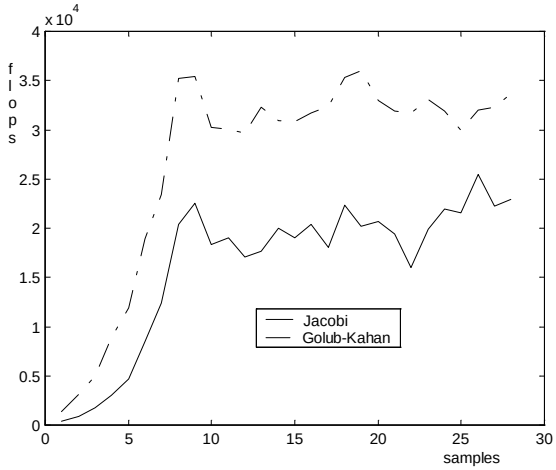

Figure 1–Simulation for M=10

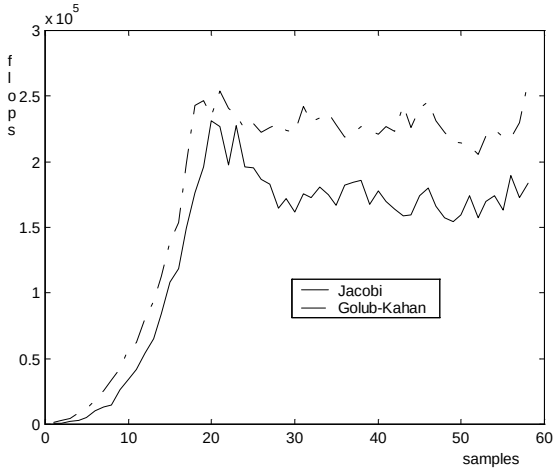

Figure 2–Simulation for M=20

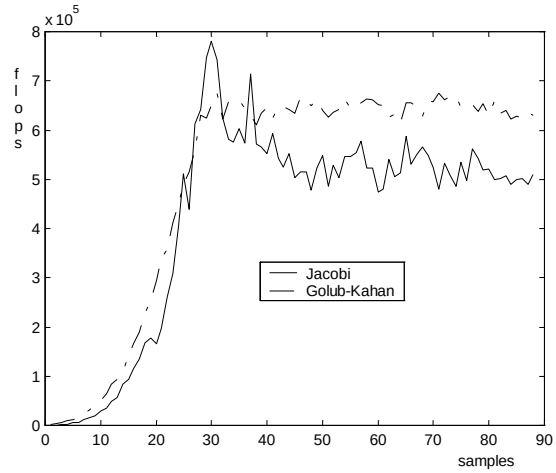

Figure 3–Simulation for M=30

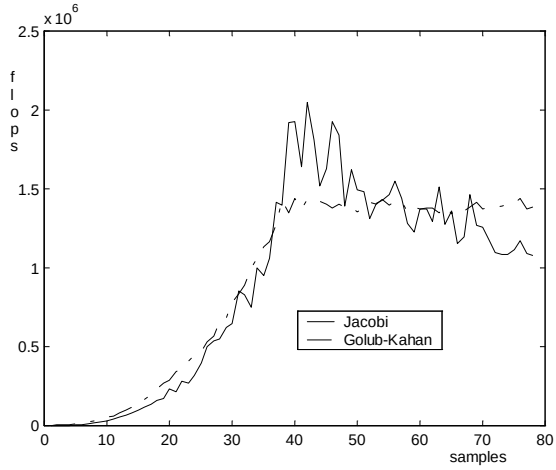

Figure 4 – Simulation for M=40

Instead of working with correlation matrices, the method is based on a new form of Data Matrix representation, such as to preserve at every time instant a desired matrix symmetry property.

The new Data Matrix is recursively built, starting from the first five data samples forming a squared, dimension 3 matrix. The Jacobi Method yields the corresponding eigenvalues-eigenvectors. As a new pair of data samples is acquired, the Data Matrix dimension increases, being the new eigenvalues-eigenvectors derived from close estimates based on the previously determined. As a limiting dimension is achieved, the algorithm is slightly modified in order to discard line and column as a new pair of line and column is incorporated to the process. This way the limiting dimension is maintained through the process.

As the previous pairs of eigenvalue-eigenvector are used as estimates for the new pairs and as they are reasonably closed to the new values, subsequent use of Jacobi Method has fast convergence, leading to computational gains with respect to conventional procedures. This is observed in four comparative cases with a Golub-Kahan step type implementation.

The method efficiency is clearly decreasing with Data Matrix dimension, being comparable and sometimes worst than Golub-Kahan step for M equal or larger than 40. It is important to notice this comparison does not take into account the evaluation steps necessary to obtain a correlation matrix. Since it works directly with the data samples, inclusion of these preparative steps would produce better comparative results for the proposed approach.

The performance improves as the final Data Matrix dimension is achieved.

VII. REFERENCES

[1] R. T. Behrens and L. L. Scharf, “Signal Processing Applications of Oblique Projection Operators”, IEEE Trans. on Signal Processing, vol. 42, no. 6, June 1994, pp. 1413-1424.

[2] R. O. Schmidt, “Multiple Emitter Location and Signal Parameter estimation”, IEEE Trans. Antennas Propagation, vol. 34, pp. 276-280, Mar 1986.

[3] G. H. Golub and C. F. V. Loan, “Matrix Computations”, The John Hopkins University Press, Baltimore, MD, third edition, 1996.

[4] S. Haykin, “Adaptive Filter Theory”, Prentice hall, Upper Side River, NJ, third edition, 1996.

[5] L Eldén and E. Sjöström, “Fast Computation of the Principal Singular Vectors of Toeplitz Matrices Arising in Exponential Data Modeling”, Signal Processing 50 (1996), pp 151-164.

[6] P. A. Pango and B. Champagne, “On the Efficient use of Givens Rotations in SVD-based Subspace Tracking Algorithms”, Preprint submitted to Elsevier Preprint, 1999.

[7] F. Vanpoucke and Marc Moonen, “Factored Spherical Subspace Tracking”, Integration, the VLSI Journal, vol. 20, Nr. 1, December 1995, pp 3-21.

[8] G. W. Stewart, “An Updating Algorithm for Subspace Tracking”, UMIACS-TR-90-86, CS-TR 2494, 1991.

[9] M. Moonen, P. Van Dooren and J. Vandewalle, “An SVD Updating Algorithm for Subspace Tracking”, SIAM Journal on Matrix Analysis and Applications, vol. 13 (1992), no. 4, pp. 1015-1038.

[10] S. D. Stearns and R. A. David, “Signal Processing Algorithms in Matlab”, Prentice hall, Upper Side River, NJ, 1996.