A Study on Parallelization of Successive Rotation Based Joint Diagonalization

Xiu-Lin Wang, Xiao-Feng Gong, Qiu-Hua Lin
School of information and communication engineering
Dalian University of Technology
Dalian City, Liaoning Province, China
xfgong@dlut.edu.cn

***Abstract*—Joint diagonalization (JD) is an instrumental tool in a vast variety of applications such as blind source separation, polarization sensitive array processing, and linear algebra based computation of tensor decompositions. Among the JD families, those based on successive rotations are a major category that minimizes the adopted highly nonlinear cost function by solving a set of simple sub-optimization problems. These sub-optimization problems are associated with certain elementary rotations that are performed over one or two rows and columns of target matrices, and thus a lower-dimensional exhaustion is required to cover and update all the matrix entries in a sequential manner. As such, the time consumed in the exhaustion procedure is in quadratic relationship with the dimensionality of target matrices and would go extremely heavy when handling large matrices. In this study, we examine and compare 3 parallelization schemes for a recently developed successive rotation based JD algorithm. The results show that these schemes can largely reduce the running time of JD with almost equal resulting accuracy when compared with the original version, when handling large matrices.**

***Keywords*—Joint diagonalization, Successive rotation, LU, Parallelization**

I. INTRODUCTION

Joint diagonalization (JD) has been successfully applied in various fields, especially in blind source separation (BSS) [1], array processing [2], and tensor factorizations [3-5]. Given a set of complex square matrices $\{\boldsymbol{C}_1,\boldsymbol{C}_2,...,\boldsymbol{C}_K\}$ with the following JD structure:

$$\boldsymbol{C}_k = \boldsymbol{A}\boldsymbol{D}_k\boldsymbol{A}^H \tag{1}$$

where $\boldsymbol{D}_k$ are diagonal, k = 1, 2,…, K, $\boldsymbol{A}$ is the loading matrix, and superscript 'H' denotes conjugated transpose, JD then seeks the unloading matrix $\boldsymbol{W} \triangleq \boldsymbol{A}^{-1}$ so that $\{\boldsymbol{W}\boldsymbol{C}_k\boldsymbol{W}^H\}_{k=1}^K$ are as diagonal as possible. In practice, the target matrices $\boldsymbol{C}_k \in \mathrm{C}^{N\times N}$ can be constructed as 3rd or 4th order cumulant slices, or 2nd order covariance matrices for different time instances or time delays [1, 6, 7].

There are many existing works devoted to JD including early ones which imposed orthogonality constraints on the loading matrix (hence the name orthogonal JD) [1], and more recent ones that considered non-orthogonal JD [7-13]. In addition, several criteria have been successively applied to JD problems including off-norm minimization, weighted least squares, and information theory based criterion, that are implemented by some specific optimization based or algebraic strategies. Among the above JD families, algebraic algorithms based on successive rotations are a major category [1, 7, 10-13]. In particular, these methods convert the highly nonlinear JD problem into a sequence of optimized elementary rotations over one or two rows and columns of target matrices, and sweep these elementary rotations to cover all the matrix entries in the sequential manner such that overall optimization is achieved. As a result of this sequential rotation based scheme, the time consumed in the exhaustion procedure is in quadratic relationship with the dimensionality of target matrices and would go extremely heavy when handling large matrices. To address this problem, parallelization of these successive rotations is considered by a few works in the open literature [14, 15].

In this paper, we consider parallelization of a recently developed LU decomposition based complex JD algorithm (LUCJD) [12]. Unlike other JD algorithms of similar kind, each elementary rotation involved in LUCJD only impacts 1 column (row) of the target matrices, which could be optimized in very simple closed-form, and this nice feature of LUCJD enables more flexible parallelization schemes in comparison to other successive rotation based algorithms. More exactly, we will introduce 3 parallelization schemes for LUCJD, and demonstrate with simulations the behaviors of these schemes with regards to both execution speed and accuracy.

II. REVIEW OF LUCJD

LUCJD is proposed in [12] as the complex-valued extension of the work in [11]. Off-norm minimization is used as criterion for LUCJD, which is defined as:

$$\tilde{\boldsymbol{W}} = \arg\min_{\boldsymbol{W}} \sum_{k=1}^{K} \mathrm{off}(\boldsymbol{W}\boldsymbol{C}_k\boldsymbol{W}^H) \tag{2}$$

where $\boldsymbol{W} \triangleq \boldsymbol{A}^{-1}$ denotes unmixing matrix, $\tilde{\boldsymbol{W}}$ is its estimate, and $\boldsymbol{C}_k$, k = 1,2,…,K, are target matrices. $\mathrm{off}(\boldsymbol{G}) = \sum_{1\le i\ne j\le N} |g_{i,j}|^2$ measures the sum of squared norms of off-diagonal entries (off-norm) for $\boldsymbol{G} \in \mathrm{C}^{N\times N}$.

This work was supported in part by Doctoral Fund of Ministry of Education of China under grant 20110041120019, and National Natural Science Foundation of China under grants 61105008, 61331019, 61379012, 61201386.

The key idea of LUCJD is to estimate unmixing matrix $\boldsymbol{W}$ with the following 2 alternating stages (till convergence to reach optimization), using LU decomposition $\boldsymbol{W} = \boldsymbol{LU}$:

$$\tilde{\boldsymbol{U}} = \arg\min_{\boldsymbol{U}} \sum_{k=1}^{K} \text{off}(\boldsymbol{U}\boldsymbol{C}_k\boldsymbol{U}^H) \tag{3.a}$$

$$\tilde{\boldsymbol{L}} = \arg\min_{\boldsymbol{L}} \sum_{k=1}^{K} \text{off}(\boldsymbol{L}\boldsymbol{C}'_k\boldsymbol{L}^H) \tag{3.b}$$

where $\boldsymbol{C}'_k = \tilde{\boldsymbol{U}}\boldsymbol{C}_k\tilde{\boldsymbol{U}}^H$ and $\tilde{\boldsymbol{W}} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}}$. In addition, both L- and U-stages are accomplished by repeating following optimized elementary rotations for all possible index pairs (i, j), $1 \le i < j \le N$, (we take U-stage for example):

$$\boldsymbol{C}_{k,new} = \boldsymbol{T}_{(i,j)}\boldsymbol{C}_{k,old}\boldsymbol{T}^H_{(i,j)}, \quad \boldsymbol{U}_{new} = \boldsymbol{T}_{(i,j)}\boldsymbol{U}_{old} \tag{4}$$

where $\boldsymbol{C}_{k,new}$ and $\boldsymbol{U}_{new}$ denote the updates of $\boldsymbol{C}_k$ and $\boldsymbol{U}$ in the current iteration, and $\boldsymbol{C}_{k,old}$ and $\boldsymbol{U}_{old}$ are the results obtained in the previous iteration, $k = 1,..., K$. $\boldsymbol{T}_{(i,j)}$ is elementary rotation matrices for U-stage, which equal the identity matrix except the upper-triangular entry $\alpha_{(i,j)}$ indexed (i, j). In particular, if we define $\beta_{i,j,p}$ and $\gamma_{i,j,p}$ as:

$$\begin{cases} \beta_{i,j,p} \triangleq \sum_{k=1}^{K}\left[\boldsymbol{C}_{k,old}(i,p)\boldsymbol{C}^*_{k,old}(j,p) + \boldsymbol{C}_{k,old}(p,j)\boldsymbol{C}^*_{k,old}(p,i)\right] \\ \gamma_{i,j,p} \triangleq \sum_{k=1}^{K}\left[\boldsymbol{C}_{k,old}(j,p)\boldsymbol{C}^*_{k,old}(j,p) + \boldsymbol{C}_{k,old}(p,j)\boldsymbol{C}^*_{k,old}(p,j)\right] \end{cases} \tag{5}$$

the optimal $\alpha_{(i,j)}$ is then given by [12]:

$$\alpha_{i,j} = -\left(\sum_{p=1,p\neq i}^{N} \gamma_{i,j,p}\right)^{-1}\left(\sum_{p=1,p\neq i}^{N} \beta_{i,j,p}\right) \tag{6}$$

The above updating procedure also holds for L-stage, with the only exception that $i > j$. A complete exhaustion of all index pairs for both L and U-stages is defined as a sweep, and there are $N(N-1)$ elementary rotations in one sweep of LUCJD, that run sequentially. As such, the number of such rotations would grow quadratically with the increase of N which results in long computational time when handling large matrices. To address this problem, we shall consider parallelization of LUCJD in the next section.

III. PARALLELIZATION OF LUCJD

To solve the above problem, a natural consideration is to implement the elementary rotations in parallel fashion, and the key to an efficient parallelization is the segmentation of entire index pairs into multiple subsets for which the optimal elementary rotation matrices could be calculated at one shot. These index pairs (i', j') and (i, j) in one subset are non-conflicting in the sense that calculation of $\alpha_{i',j'}$ relies little on the entries of target matrices that are updated by $\boldsymbol{T}_{(i,j)}$. In addition, noting that the (i, j)th elementary rotation in U-stage would impact the ith row and column of the target matrices, we could develop the following 3 parallelization schemes (in the following subsections we only address the U-stage, same reasoning holds for L-stage).

A. Column-wise Parallelization

The subsets in this scheme consist of index pairs with equal column index j. More exactly, the jth subset is defined as $\Omega^C_j \triangleq \{(i, j)\,|\,i = 1, 2,..., j-1\}$, $j = 2, 3,..., N$, and elementary rotation matrices associated with index pairs $(i, j) \in \Omega^C_j$ are calculated simultaneously. We note here that only a partial portion in the calculation of $\alpha_{i,j}$ is dependent on other index pairs from the same subset to reach as much as possible non-conflicting requirement for subsets in parallelization. More exactly, (6) could be rewritten as:

$$\alpha_{i,j} = -\left(\sum_{p=j}^{N}\gamma_{i,j,p} + \sum_{p=1,p\neq i}^{j-1}\gamma_{i,j,p}\right)^{-1}\left(\sum_{p=j}^{N}\beta_{i,j,p} + \sum_{p=1,p\neq i}^{j-1}\beta_{i,j,p}\right) \tag{7}$$

where terms $\sum_{p=j}^{N}\gamma_{i,j,p}$ and $\sum_{p=j}^{N}\beta_{i,j,p}$ are independent of other index pairs in Ω^C_j, j = 2,3,…, N. As a result, the main difference with regards to performance between column-wise parallelized LUCJD and its original sequential version lies in the contributions of $\sum_{p=1,p\neq i}^{j-1}\gamma_{i,j,p}$ and $\sum_{p=1,p\neq i}^{j-1}\beta_{i,j,p}$ that are dependent on other index pairs. In particular, the portion of these 2 terms is very little when j is small, and would go up as j increases. On the other hand, by using this parallelization scheme, only the 1-dimensional exhaustion over the j index is required for each sweep of LUCJD, and thus the execution time would be much reduced (in linear relations with the increase of N) compared with its original sequential version (in quadratic relations with N).

B. Row-wise Parallelization

A natural counterpart to the above column-wise scheme is the row-wise one. Subsets are constructed by combining elementary rotation matrices associated with index pairs having equal i index (row index): $\Omega^R_i \triangleq \{(i, j)\,|\,j = i+1,..., N\}$, $i = 1, 2,..., N-1$, and these matrices belonging to the same subset are optimized simultaneously. Similar to the column-wise scheme, the original 2-dimensional exhaustion over both i and j indices are reduced to the 1-dimensional one, and thus significant reduction of running time could be expected. However, it is important to note, when compared with the column-wise parallelization scheme, that the subset elements in the row-wise scheme are not non-conflicting and might result in performance loss. More precisely, with index i fixed and j distinct, all rotation matrices in the ith subset would impact the ith row and thus the terms $\gamma_{i,j,p}$ and $\beta_{i,j,p}$ in (6) are dependent on (hence conflicting with) other index pairs in the ith subset.

C. Diagonal-wise Parallelization

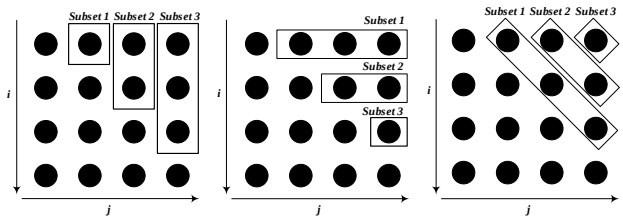
Comparing the above column-wise and row-wise schemes, we note that subsets containing index pairs with distinct i indices might be more favorable for parallelization as the subset elements are more non-conflicting in such case. Therefore, we could consider grouping index pairs along the left-to-right descending diagonals as subsets: $\Omega^D_j \triangleq \{(i, i+j)\,|\,i = 1, 2,..., N-j\}$, $j = 1, 2,...N-1$, and calculate the

optimal rotation matrices associated with these index pairs at one slot. We note that the non-conflicting requirement for elementary rotation matrices in one subset is partially fulfilled similar to the column-wise scheme, if we rewrite $\alpha_{i,i+j} \in \Omega_j^D$ (6) in the following form:

$$\alpha_{i,i+j} = -\frac{\sum_{p=i+j}^{N} \beta_{i,i+j,p} + \sum_{p=1,p\neq i}^{i+j-1} \beta_{i,i+j,p}}{\sum_{p=i+j}^{N} \gamma_{i,i+j,p} + \sum_{p=1,p\neq i}^{i+j-1} \gamma_{i,i+j,p}} \quad (8)$$

where terms $\sum_{p=i+j}^{N} \beta_{i,i+j,p}$ and $\sum_{p=i+j}^{N} \gamma_{i,i+j,p}$ are independent of other index pairs in Ω_j^D, j = 1,3,…, N-1, and thus its main difference with the original sequential LUCJD lies in the contributions of $\sum_{p=1,p\neq i}^{i+j-1} \beta_{i,i+j,p}$ and $\sum_{p=1,p\neq i}^{i+j-1} \gamma_{i,i+j,p}$ that are dependent on other index pairs. In particular, the portion of these 2 terms is large when j is small, and would go down as j increases. In addition, similar to the above 2 parallelization schemes, the execution time would be largely reduced compared with its original sequential version. It is important to note that this scheme is in principle analogous to the one proposed in [14], in the point that the index pairs in one subsets contain distinct i indices and j indices. The reason for the work in [14] to let both i and j be different within one subset is that the Givens rotations therein impact 2 rows and 2 columns each time. However, as is afore-mentioned, we only need to let the i indices be different within one subset for LUCJD as the elementary rotation matrices only impact 1 row and 1 column each time. As such, the particular use of LUCJD would enable more flexible parallelization schemes than other JD methods of similar kind.

The subset segmentation for the above 3 parallelization schemes is visualized in Figure 1.



(a) Column-wise (b) Row-wise (c) Diagonal-wise

Figure 1. Subset segmentation for parallelization of LUCJD

IV. Simulation Results

In this section, we illustrate and compare the performances of the introduced 3 parallelization schemes for LUCJD in comparison with the original sequential version, as well as the tournament player's ordering (TPO) parallelization scheme introduced in [14] using simulations. The target matrices are generated as:

$$\boldsymbol{C}_k = P_s \boldsymbol{A} \boldsymbol{D}_k \boldsymbol{A}^H + P_n \boldsymbol{N}_k, \quad k = 1,2,...,K \quad (9)$$

where $\boldsymbol{D}_k \in \mathrm{C}^{N\times N}$ are diagonal matrices , $\boldsymbol{A} \in C^{N\times N}$ is the loading matrix and $\boldsymbol{N}_k \in \mathrm{C}^{N\times N}$ is white Gaussian noise term with zero mean and unit variance. K and N denote the number and the dimensionality of target matrices. P_s and P_n are signal level, and noise level, respectively. Signal-to-noise ratio (SNR) is defined as:

$$SNR = 10\log_{10}(P_s / P_n) \quad (10)$$

Moreover, we use performance index (PI) to evaluate the JD qualities of compared algorithms, which is defined as:

$$PI = \frac{1}{2N(N-1)}\left[\sum_{i=1}^{N}\left(\sum_{j=1}^{N}\frac{|g_{ij}|}{\max_k |g_{ik}|} - 1\right) + \sum_{i=1}^{N}\left(\sum_{j=1}^{N}\frac{|g_{ji}|}{\max_k |g_{ki}|} - 1\right)\right] \quad (11)$$

where g_{ij} is the $(i, j)th$ entry of $\boldsymbol{G} = \tilde{\boldsymbol{W}}\boldsymbol{A}$, $\tilde{\boldsymbol{W}}$ denotes the estimated unloading matrix and $\boldsymbol{A}$ is the true loading matrix.

The following simulations are done under the following configurations; CPU: Intel Core i7 2.93Hz; Memory: 16GB; System: 64bit Windows 7; Matlab R2010b.

A. Simulation 1 – Convergence Pattern

The 1st simulation compares the convergence patterns of all competitors. We generate 10 target matrices of size 10×10 $K = N = 10$, fix SNR to 20dB, and draw PI curves versus number of sweeps from 5 independent runs in Figure 2.

We note that the parallelization schemes behave almost identically to the original sequential LUCJD, with only 1 or 2 more sweeps at times. This indicates that using parallelization for LUCJD has little impact on the converging behaviors.

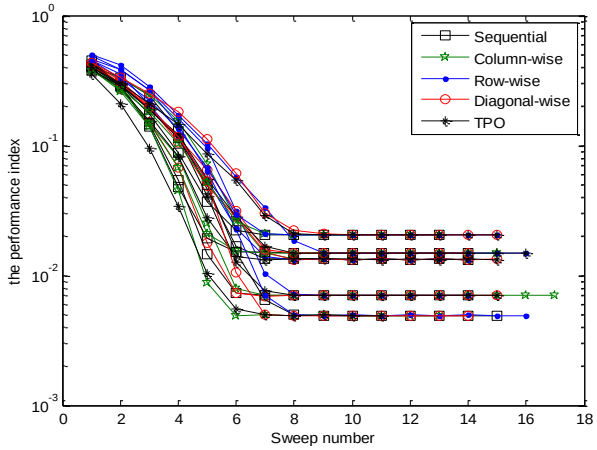


Figure 2. PI versus number of iterations, N = 10, K = 10, SNR = 20dB

B. Simulation 2 – Execution Time

In the 2nd simulation, execution times of all the 5 compared algorithms are measured against the matrix dimensionality N that are varied from 10 to 50. We set the number of target matrices K to 20 and SNR to 20dB, respectively, and illustrate in Figure 3 the average running time curves acquired from 100 Monte Carlo runs.

From Figure 3, we can see that using parallelization for LUCJD could largely reduce the execution time when compared with the sequential version. Furthermore, we note that this merit would become more significant as N increases, indicating that parallelization is more favorable for handling

high dimensional datasets. This observation is in coincidence with our analysis in section 3, that using parallelization would much reduce the execution time in aspects that its relationship with N is improved from being quadratic to being approximately linear.

Moreover, among the 4 compared parallelization schemes, we note that the proposed row-wise scheme is the fastest, the column-wise and TPO schemes ranked 2nd and 3rd fastest, respectively, yet the diagonal-wise one is slightly inferior.

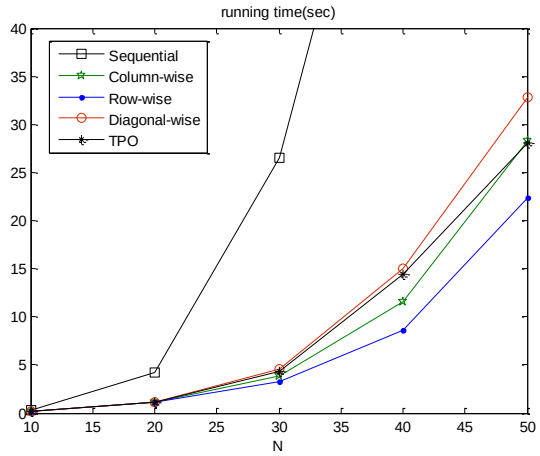


Figure 3. Average running time versus dimensionality, K = 20, SNR = 20dB

C. *Simulation 3 – Joint Diagonalization Quality*

In this simulation, we illustrate the performances of the competitors by means of PI values against SNR. We fix K = 20, N = 30, let SNR vary from 0-30dB, and draw PI curves obtained from 100 Monte Carlo runs in Figure 4.

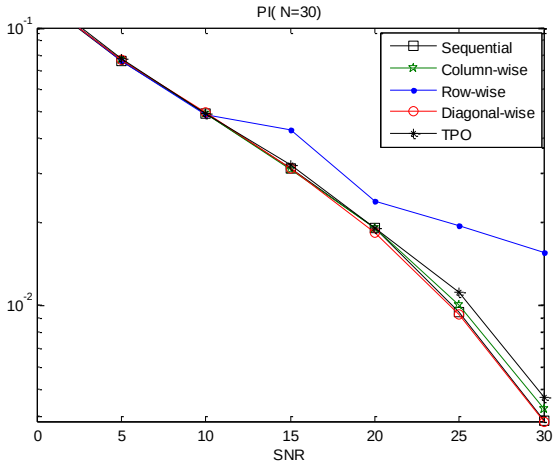


Figure 4. PI versus SNR, N = 30, K = 20

From Figure 4 we note that all the compared schemes yield equal performance for relatively low SNR (0-10dB), and start to behave diversely when SNR goes beyond. In particular, performance of row-wise scheme is inferior to all the others, and this is in accordance with our analysis in subsection 3.B, that subset elements in row-wise scheme are less non-conflicting which is likely to result in performance loss. In addition, we note that the diagonal-wise scheme performs almost identical to the original sequential one, closely followed by column-wise and TPO schemes. The above observations suggest that LUCJD with various parallelization schemes would perform almost equally well as the original sequential version.

V. CONCLUSION

Joint diagonalization (JD) is an instrumental tool for array signal processing and computation of tensor factorizations. This paper addresses the problem of parallelization for JD with successive rotations for acceleration, and introduces 3 schemes for a recently developed LU decomposition based complex JD algorithm (LUCJD), termed as column-wise, row-wise, diagonal-wise schemes, respectively. Simulations are conducted to compare these 3 schemes with the original sequential LUCJD, and the existing tournament player's ordering (TPO) parallelization scheme, with regards to both execution time and JD quality. Results generally show that using the proposed parallelization schemes would largely reduce the running time without losing the JD quality. In particular, the row-wise scheme generates fastest yet least accurate calculation, and diagonal-wise scheme gives the best JD quality (almost equal to the sequential version) yet slightly less efficiency (still much faster than the sequential one) than other parallelization schemes. The column-wise and TPO schemes perform somewhere in the middle of the 4 compared parallelization schemes, with slight superiority from the former over the latter with regards to both speed and accuracy.

REFERENCES

[1] J. -F. Cardoso and A. Souloumiac, "Blind beamforming for non-Gaussian signals," *Proc. IEE.-F*, vol. 140, no. 6, pp. 362-370, 1993.

[2] X. F. Gong, K. Wang, Q. H. Lin, Z. W. Liu, Y. G. Xu, "Simultaneous source localization and polarization estimation via non-orthogonal joint diagonalization with vector-sensors," *Sensors.* Vol. 12, no. 3, pp. 3394-3417, 2012.

[3] L. Delathauwer, "A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization," *SIAM J.Matrix Anal. Appl.*, vol.28, no.3, pp. 642-666, 2006.

[4] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms-PART II definitions and uniqueness," *SIAM. J. Matrix Anal. Appl.*, vol 30, no 3, pp.1033-1066, 2008.

[5] X. F. Gong, Y. N. Hao, Q. H. Lin, "Joint canonical polyadic decomposition of two tensors with one shared loading matrix," in *Proc. MLSP2013*, Southampton, U.K., Sep. 22-25, 2013.

[6] A. Belouchrani, K. Abed-Meraim, J. -F. Cardoso, and E. Moulines, "A blind source separation technique using second-order statistics," *IEEE Trans. Signal Process.*, vol. 45, no. 2, pp. 434-444, Feb. 1997.

[7] D. T. Pham, "Joint approximate diagonalization of positive definite matrices," *SIAM J. Matrix Anal. Appl.*, vol.23, no. 4, pp. 1136-1152, Apr. 2001.

[8] P. Tichavsky and A. Yeredor, "Fast approximate joint diagonalization incorporating weight matrices", *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 878-891, Mar. 2009.

[9] X. L. Li, X. D. Zhang, "Nonorthogonal joint diagonalization free of degenerate solution," IEEE Trans. Signal Process., vol. 55, no. 5, pp. 1803-1814, May. 2007.

[10] A. Souloumiac, "Nonorthogonal joint diagonalization by combining givens and hyperbolic rotations," *IEEE Trans. Signal Process.*, vol. 57, no. 6, pp. 2222-2231, Jun. 2009.

[11] B. Afsari, "Simple LU and QR based non-orthogonal matrix joint diagonalization," *Proc. ICA'2006*, Charleston, USA, pp. 1-7, Mar. 5-8, 2006.

[12] K. Wang, X. F. Gong, and Q. H. Lin, "Complex non-orthogonal joint diagonalization based on LU and LQ decompositions," in *Proc. LVA/ICA*, Tel Aviv, Israel, pp. 50-57, Mar. 12-15, 2012.

[13] A. Mesloub, K. Abeb-Meraim, A. Belouchrani, "A new algorithm for complex non orthogonal joint diagonalization based on Shear and Givens rotations," *IEEE Trans. Signal Process.*, vol. 62, no. 8, pp. 1913-1925, Apr. 2014.

[14] A. Holobar, M. Ojstersek, D. Zazula, "A new approach to parallel joint diagonalization of symmetric matrices," in *Proc. EUROCON2003*, Ljubljana, Slovenia, Sep. 22-24. 2003.

[15] S. Singer, S. Singer, V. Novaković, A. Uščumlić, V. Dunjko, "Novel modifications of parallel Jacobi algorithms," *Numer Algorithms.*, vol. 59, no. 1, pp. 1-27, Jan. 2012.