A Jacobi Method for Signal Subspace Computation

St. Paul[a] and J. Götze[b]

[a]Institute for Network Theory and Circuit Design
TU Munich, D-80290 Munich, Germany

[b]Computer Engineering Institute
University of Dortmund, Germany

ABSTRACT

The Jacobi method for singular value decomposition is well-suited for parallel architectures. Its application to signal subspace computations is well known. Basically the subspace spanned by singular vectors of large singular values are separated from subspace spanned by those of small singular values. The Jacobi algorithm computes the singular values and the corresponding vectors in random order. This requires sorting the result after convergence of the algorithm to select the signal subspace.

A modification of the Jacobi method based on a linear objective function merges the sorting into the SVD-algorithm at little extra cost. In fact, the complexity of the diagonal processor cells in a triangular array get slightly larger. In this paper we present these extensions, in particular the modified algorithm for computing the rotation angles and give an example of its usefulness for subspace separation.

Keywords: Subspace methods, Singular value decomposition, Jacobi method, parallel architecture, optimization on manifolds, Lie groups

1. INTRODUCTION

The singular value decomposition (SVD) of a $m \times n$ ($m \geq n$) matrix $\boldsymbol{X}$ is defined as:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices composed of the left sided and right sided singular vectors, respectively, and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n, 0_{n\times(m-n)})$ is a diagonal matrix containing the singular values (σ_i: i–th singular value). Computing the SVD is a widely encountered problem in many signal processing applications (e.g. parameter estimation, system identification).[1,18] In general, there are two methods for computing the SVD[17]: (1) QR–algorithm and (2) Jacobi (Kogbetliantz) method (we omit Arnoldi process based methods, here). The QR–algorithm with preparatory bidiagonalization is computationally much more efficient than the Jacobi method. On the other hand, however, Jacobi methods exhibit much more inherent parallelism than the QR–algorithm[4] and yield the singular values with higher numerical accuracy.[15] Therefore, in real time signal processing applications parallel implementations of Jacobi methods are widely used.[1] Similar techniques were considered in[2] but there a ordinary differential equation was discretized and iterated.

In contrary to the QR–algorithm which yields the singular values in descending order ($\sigma_i \geq \sigma_j$, $i > j$), the Jacobi method yields the singular values in arbitrary order, i.e., it is not known in advance in which matrix position certain singular values and their associated singular vectors can be found. In general, the ordering of the singular values is not a crucial criteria. When it comes to applications, however, singular values and singular vectors have a certain meaning. In signal subspace estimation, the singular vectors of the large singular values span the signal subspace. For this reason after the computation of the SVD by a Jacobi type method a selection algorithm has to be performed

Other author information: (Send correspondence to S.P.)
S. P.: Email: stpa@nws.e-technik.tu-muenchen.de.
J. G.: Email: jugo@carla.e-technik.uni-dortmund.de.

in order to find the base (singular vectors) of the signal subspace. This selection, or partial sorting is not a cheap operation in a parallel environment. In tracking applications where only partial sweeps are performed, the on–line selection gets even more important.

Computing the SVD can be considered an optimization process on a differential manifold.[3] Usually, the optimization (minimization) is done with respect to the off-diagonal norm. There is another simple objective function, however, namely trace($\boldsymbol{N}^T\boldsymbol{X}$), where $\boldsymbol{N}$ is a diagonal matrix of suitable dimension. Taking this objective function one gets the singular values in the same order as the entries of $\boldsymbol{N}$ (i.e. choosing $n_{ii} > n_{jj}$ for $i < j$ the Jacobi method yields $\sigma_i > \sigma_j$). Thereby, a Jacobi method is obtained which enables a choosen ordering of the singular values/vectors.

Here the parallel implementation of the Jacobi method based on the new optimization criteria trace($\boldsymbol{N}^T\boldsymbol{X}$) is discussed. The resulting algorithm can be implemented on a parallel Jacobi array based on the cyclic–by–row ordering scheme[4] with a slightly higher complexity of the diagonal cells (essentially they have to determine if inner or outer rotations are used[19]). More specifically, the diagonal entries of $\boldsymbol{N}$ have to be stored in these processors. In each processor, the two entries of n_{ii}, n_{jj} control the rotation angle such that the singular values are sorted according to a chosen ordering of $\boldsymbol{N}$. The n_{ii} are moved between the processors according to the parallel ordering scheme of the Jacobi method. Hence, it is always known in what processor certain singular values can be found. Since parallel Jacobi schemes are usually cyclic, the k largest singular values can be found in the first k processors (for the above chosen $\boldsymbol{N}$). The same holds for the singular vectors (first k columns of $\boldsymbol{V}$). By slightly modifying (based on a modified objective function) the tasks of the diagonal processors one is able to combine the parallelism of Jacobi methods with the identification of certain singular values/vectors.

In section 2. the Jacobi method is considered as an optimization process on a differential manifold. Furthermore, the simple objective function which yields the additional sorting feature of the Jacobi method is introduced. The parallel implementation of the Jacobi method is discussed and the required modifications in order to include the sorting of the singular values/vectors are shown. In section 3. Jacobi methods are applied to signal subspace methods for spectral estimation. The simulations show the advantages of the sorting Jacobi method with respect to a simple identification of the singular vectors spanning the signal subspace. Particularly, for on–line computations of the signal subspace in time–varying environments the sorting is essential for the integration of the entire system architecture.

2. A SORTING JACOBI SVD METHOD AND ITS PARALLEL IMPLEMENTATION

The application of Jacobi methods for signal subspace computations require a separation of large and small singular values and its right singular vectors. The Jacobi algorithm, performed on a serial or parallel architecture doesn't provide any information on the position of the large singular values after convergence. To extract the singular vectors, a sorting operation is needed. Regardless of their complexity, one has to spend additional hardware.

Recall that the computation of the rotation angles to minimize the off-diagonal norm requires the solution of a quadratic equation if the transformation is composed of 2×2 plane rotations. Clearly, there are two solutions.

To add the feature of sorting to the Jacobi-method, two approaches are possible:

1. rotation strictly in one direction (see below) while minimizing the off-diagonal norm
2. modification of the objective function to inherently sort the data.

Both approaches lead to almost the same algorithm but only the second one allows a proof of convergence. For the first method, the rotation angles is not uniquely determined. and examples for of rotation angles sequences where the algorithm does n't converge are well known.[5]

For the analysis of SVD-algorithms, methods from differential geometry are helpful. Let $\boldsymbol{\Sigma}, \boldsymbol{N} \in \mathcal{R}^{m\times n}$ $(m \geq n)$ be rectangular matrices

$$\boldsymbol{N} = \mathrm{diag}(\mu_1, \ldots, \mu_n, 0_{n\times(m-n)})$$

$$\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n, 0_{n\times(m-n)})$$

wit $\mu_1 > \ldots > \mu_n > 0$ and $\sigma_1 > \ldots > \sigma_n \geq 0$. The Lie groups $\mathrm{O}(n) \times \mathrm{O}(m)$ act on these matrices by

$$(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{X}) \mapsto \boldsymbol{U}\boldsymbol{X}\boldsymbol{V}^T.$$

The set of matrices with fixed singular values

$$M(\boldsymbol{\Sigma}) = \{\boldsymbol{U}\boldsymbol{X}\boldsymbol{V}^T | \boldsymbol{U} \in \mathrm{O}(n), \boldsymbol{V} \in \mathrm{O}(m), m \geq n\}$$

is an orbit of the group action and therefore a compact smooth manifold. Thus the computation of singular values amounts to the construction of an objective function which has as minimum a diagonal matrix with the singular values.

Algorithms for computing singular values or eigenvalues leave the Frobenius norm of the matrix invariant. Based on this fact, the minimization of the off-diagonal norm

$$f_q(\boldsymbol{X}) = \mathrm{trace}(\boldsymbol{X}^T\boldsymbol{X}) - \mathrm{trace}(\mathrm{diag}(\boldsymbol{X}^T)\mathrm{diag}(\boldsymbol{X})) \tag{1}$$

during the computation by a sequence of orthogonal rotations (grouped together in sweeps) brings the matrix into diagonal form (apart from nongeneric cases). The departure from the diagonal form is measured by a quadratic (!) function in the x_{ij} $(i \neq j)$.

Example: Given a 2×2 matrix $(d_1 \neq d_2)$

$$\boldsymbol{X} = \begin{pmatrix} d_1 & \\ & d_2 \end{pmatrix}.$$

This matrix constitutes one minimum of the above defined objective function f_q but it is not the only solution because the permuted diagonal entries $\boldsymbol{X} = \mathrm{diag}(d_2, d_1)$ satisfy the minimum condition as well.

The objective function (1) is invariant under permutation. To show this we restrict to the case of a symmetric matrix. Consider a diagonal matrix $\boldsymbol{D}$ and an orthogonal transformation $\boldsymbol{Q}$. For the transformed matrix $\boldsymbol{F} = \boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}$ the function f_q reads:

$$\begin{aligned} f(\boldsymbol{F}) &= \mathrm{trace}(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q}) - \mathrm{trace}(\mathrm{diag}(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q})\mathrm{diag}(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q})) \\ &= \mathrm{trace}(\boldsymbol{D}^T\boldsymbol{D}) - \mathrm{trace}(\mathrm{diag}(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q})\mathrm{diag}(\boldsymbol{Q}^T\boldsymbol{D}\boldsymbol{Q})). \end{aligned}$$

Taking a diagonal matrix (as matrix operation) and orthogonal similarity transformation commute if and only if the similarity transformation maps $\boldsymbol{D}$ onto a diagonal matrix. Such an orthogonal transformation has to be a permutation, that is, the objective function (1) is invariant under permutation. The fixed point of the minimization procedure is not unique. For distinct singular values $\min(m, n)!$ different fixed points exist.

A unique minimum (multiple singular/eigenvalues excluded) will be obtained for a linear objective function which was proposed by Brockett some time ago,[63] $(k = \mathrm{const})$:

$$f_l(\boldsymbol{X}) = \|\boldsymbol{X} - \boldsymbol{N}\|_2^2 = \mathrm{trace}((\boldsymbol{X} - \boldsymbol{N})^T(\boldsymbol{X} - \boldsymbol{N})) = -2\mathrm{trace}(\boldsymbol{N}^T\boldsymbol{X}) + k. \tag{2}$$

This function measures the distance between the given data matrix and the prescribed matrix $\boldsymbol{N}$ and leads naturally to a linear objective function. This is true because the terms $\mathrm{trace}(\boldsymbol{X}^T\boldsymbol{X})$ and $\mathrm{trace}(\boldsymbol{N}^T\boldsymbol{N})$ remain fixed under orthogonal transformations. Since the function f_l has to be minimized, the trace term is maximized. The function f_l is minimal, if and only if the entries of the diagonalized matrix $\boldsymbol{X}$ are sorted similar to $\boldsymbol{N}$. $\boldsymbol{N}$ controls the ordering relation.

The convergence of the SVD-algorithm for the objective function (2) was shown in[7] and more[8] by methods of global analysis and differential geometry.

It is noteworthy that the application of $\mathrm{SO}(n)$ transformations only necessarily violates the positivity of the singular values (consider the sign of the determinant). The sign of one singular value has to be free to cover negative determinants.

A matrix $\boldsymbol{X}$ is a critical point (unique global maximum) of the objective function (2) if and only if

$$\boldsymbol{N}\boldsymbol{X}^T = \boldsymbol{X}\boldsymbol{N}^T \quad \text{and} \quad \boldsymbol{N}^T\boldsymbol{X} = \boldsymbol{X}^T\boldsymbol{N}. \tag{3}$$

The global maximum is characterized by

$$\boldsymbol{X} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n, 0_{n\times(n-m)}) \qquad \text{for } \det(\boldsymbol{N}^T\boldsymbol{X}) = 0 \tag{4}$$

$$\boldsymbol{X} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \qquad \text{for } \det(\boldsymbol{N}^T\boldsymbol{X}) > 0 \tag{5}$$

$$\boldsymbol{X} = \operatorname{diag}(\sigma_1, \ldots, -\sigma_n,) \qquad \text{for } \det(\boldsymbol{N}^T\boldsymbol{X}) < 0 \tag{6}$$

and $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n \geq 0$. The singular values are sorted.

To show the sorting property one considers a single transformation $\operatorname{trace}(\boldsymbol{N}^T \mathrm{e}^{t\Omega}\boldsymbol{X}\mathrm{e}^{-t\Pi})$ where Ω and Π are the generators of the Lie group SO(2) embedded SO(n), SO(m) respectively. This can be done because the manifold is smooth. The vanishing first derivative with respect to t yields its value for a critical point of the function and by the second derivative one can check for minimum or maximum. This check is performed for all generators of the orthogonal group, i.e., for each rotation direction (the generators span a linear space).

Example: Consider $\boldsymbol{X}_d = \begin{pmatrix} x_1 & \\ 0 & x_2 \end{pmatrix}$, which is a critical point of the objective function f_l. This can be seen from

$$\frac{\mathrm{d}}{\mathrm{d}t} f_l|_{t=0} = \frac{\mathrm{d}}{\mathrm{d}t}\operatorname{trace}(\boldsymbol{N}^T \mathrm{e}^{t\Omega}\boldsymbol{X}_d\mathrm{e}^{-t\Pi})|_{t=0} = \operatorname{trace}(\boldsymbol{N}^T(\boldsymbol{\Omega}\boldsymbol{X}_d - \boldsymbol{X}_d\boldsymbol{\Pi})) = 0$$

($\boldsymbol{\Omega} = \omega \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, $\boldsymbol{\Pi} = \pi \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$) . For the second derivative holds

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}(-f_l)|_{t=0} = \operatorname{trace}(\boldsymbol{N}^T(\boldsymbol{\Omega}^2\boldsymbol{X}_d + \boldsymbol{X}_d\boldsymbol{\Pi}^2 - 2\boldsymbol{\Omega}\boldsymbol{X}_d\boldsymbol{\Pi})) = -\frac{1}{2}((\omega-\pi)^2(n_1+n_2)(x_1+x_2) + (\omega+\pi)^2(n_1-n_2)(x_1-x_2)).$$

This term is negative if and only if $x_1 > x_2$ with the above choice of $\boldsymbol{N}$. Therefore f_l is minimal (note the sign convention).

The algorithm for computing the singular values by the Jacobi methods is shown in Figure 1. It splits in 2 steps as opposed to 3 steps with the off-norm objective function (sorting as 3rd step).

Some comments are in order

- The sign correction by $\boldsymbol{D}$ is necessary to cover the case (6). Note, that it is also possible to compute the QRD such that $\boldsymbol{R}_{i,i} > 0$ already holds.[14] The negative sign of the determinant is provided by $\boldsymbol{Q}$. Otherwise special orthogonal rotations do not suffice.
- The function *rotsort* is standard apart from the double framed lines of code. These lines have to be added for sorting the singular values. Since the data are moved in the processor field, so does the diagonal of $\boldsymbol{N}$. Note that only the diagonal of $\boldsymbol{N}$ is really needed. The matrix notation is applied in the algorithm to clarify the computations. The values $\boldsymbol{N}_{ij}$ can be chosen almost arbitrarily (apart form being positive and sorted) when starting the algorithm. In particular, its values can be tailored to hardware needs (e.g. special binary representation).
- It is also possible to avoid the QR-decomposition by performing 2×2 rotations/reflections (cf.[7]) dirctly on the matrix $\boldsymbol{X}$.
- The algorithm 6.2.1 – 6.2.3 in[8] for singular value decomposition is based on the trace function maximization. However the ordering control matrix $\boldsymbol{N}$ doesn't show up explicitly but is incorporated in the selection of the correct rotation angle. This is not needed in the case discussed there, because all rotations are performed such that the large entries move to left upper corner. Due to the data transport on a parallel architecture the order relation of the singular values is time-variant. Therefore the diagonal processors need further information for the angle determination. Otherwise convergence is not possible.

Algorithm to decompose $\boldsymbol{X} \in \mathcal{R}^{m\times n}$ $(m \geq n)$

1. QR-decomposition
2. Compute $\boldsymbol{D}$ in $\boldsymbol{QDD}^{-1}\boldsymbol{R}$ such that $\det(\boldsymbol{R}(1:n, 1:n)) > 0$
3. Diaognalize $\boldsymbol{R}(1:n, 1:n)$ by a cyclic scheme, e.g. by Brent and Luk,[4] operating on $2\times$ problems. Such a optimization problem is solvbed by the algorithm *rotsort*.

function$[\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{Y}, \boldsymbol{N}] = \text{rotsort}(\boldsymbol{X}, \boldsymbol{N})$;

$\boldsymbol{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

$x_1 = (\boldsymbol{X}_{22} + \boldsymbol{X}_{11})/2$;

$x_2 = (\boldsymbol{X}_{22} - \boldsymbol{X}_{11})/2$;

$y_1 = (\boldsymbol{X}_{21} - \boldsymbol{X}_{12})/2$;

$y_2 = (\boldsymbol{X}_{21} + \boldsymbol{X}_{12})/2$;

$[W_1, \boldsymbol{G}_1] = \text{givrot}([x_1; y_1])$;

$[W_2, \boldsymbol{G}_2] = \text{givrot}([x_2; y_2])$;

$w_1 = \arctan(\boldsymbol{G}_{1,21}/\boldsymbol{G}_{1,11})$;

$w_2 = \arctan(\boldsymbol{G}_{2,21}/\boldsymbol{G}_{2,11})$;

$\boldsymbol{J}_1 = \begin{pmatrix} \cos(w_1/2) & -\sin(w_1/2) \\ \sin(w_1/2) & \cos(w_1/2) \end{pmatrix}$

$\boldsymbol{J}_2 = \begin{pmatrix} \cos(w_2/2) & -\sin(w_2/2) \\ \sin(w_2/2) & \cos(w_2/2) \end{pmatrix}$

$\boldsymbol{U} = \boldsymbol{J}_1\boldsymbol{J}_2^T$

$\boldsymbol{V} = \boldsymbol{J}_1^T\boldsymbol{J}_2$

$\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{X}\boldsymbol{V}^T$

If $(\boldsymbol{N}_{11} - \boldsymbol{N}_{22})(\boldsymbol{Y}_{11} - \boldsymbol{Y}_{22}) < 0)$

$\boldsymbol{U} = \boldsymbol{P}\boldsymbol{U}$

$\boldsymbol{V} = \boldsymbol{P}\boldsymbol{V}$

end

% reflexion for parallel implementation

$\boldsymbol{U} = \boldsymbol{P}\boldsymbol{U}$

$\boldsymbol{V} = \boldsymbol{P}\boldsymbol{V}$

$\boldsymbol{Y} = \boldsymbol{P}\boldsymbol{Y}\boldsymbol{P}$

$\boldsymbol{N} = \boldsymbol{P}\boldsymbol{N}\boldsymbol{P}$

end

Figure 1. Algorithm for SVD-Jacobi method and the diagonal processors of the triangular array.

- The algorithm *rotsort* only describes the computations of the diagonal cells for a general purpose computer. Practical implementations would use hardware specific implementations of the rotation computations, e.g., the CORDIC algorithm as in.[16,13] This does not effect, however, the specific computations required for the sorting feature.

The triangular array Brent and Luk[4] can be utilized for parallel implementation of the SVD-Jacobi algorithm. The preparatory QR-decomposition can be performed on the same array.[9]

On a parallel architecture the data are moved around in the processor field. However, the index sequence of the rotations is fixed and known. In the triangular array of Brent an Luk the the rotations are determined in the diagonal processors. These rotations correspond to rotations in the first subdiagonal. The index scheme guarantees that each off-diagonal entry is moved into either of these matrix positions exactly once.

All schemes are cyclic. Thereby, the matrix $\boldsymbol{N}$ appears in the same ordering as the initialization after a known number of sweeps.

To illustrate the procedure we consider a matrix containing the indices as entries and don't perform any rotations but transport the data only. Then, the diagonal processor cells permute the data as shown in (7):

$$\begin{pmatrix} 11 & 12 & 13 & 14 & 15 & 16 \\ 21 & 22 & 23 & 24 & 25 & 26 \\ 31 & 32 & 33 & 34 & 35 & 36 \\ 41 & 42 & 43 & 44 & 45 & 46 \\ 51 & 52 & 53 & 54 & 55 & 56 \\ 61 & 62 & 63 & 64 & 65 & 66 \end{pmatrix} \rightarrow \begin{pmatrix} 22 & 24 & 21 & 26 & 23 & 25 \\ 42 & 44 & 41 & 46 & 43 & 45 \\ 12 & 14 & 11 & 16 & 13 & 15 \\ 62 & 64 & 61 & 66 & 63 & 65 \\ 32 & 34 & 31 & 36 & 33 & 35 \\ 52 & 54 & 51 & 56 & 53 & 55 \end{pmatrix} \rightarrow$$
$$\begin{pmatrix} 44 & 46 & 42 & 45 & 41 & 43 \\ 64 & 66 & 62 & 65 & 61 & 63 \\ 24 & 26 & 22 & 25 & 21 & 23 \\ 54 & 56 & 52 & 55 & 51 & 53 \\ 14 & 16 & 12 & 15 & 11 & 13 \\ 34 & 36 & 32 & 35 & 31 & 33 \end{pmatrix} \rightarrow \begin{pmatrix} 66 & 65 & 64 & 63 & 62 & 61 \\ 56 & 55 & 54 & 53 & 52 & 51 \\ 46 & 45 & 44 & 43 & 42 & 41 \\ 36 & 35 & 34 & 33 & 32 & 31 \\ 26 & 25 & 24 & 23 & 22 & 21 \\ 16 & 15 & 14 & 13 & 12 & 11 \end{pmatrix}. \tag{7}$$

An orthogonal matrix $\boldsymbol{Q} \in \mathrm{O}(n)$ is generated by $\frac{n(n-1)}{2}$ elementary rotations (parameters). The processor field computes $n-1$ (n matrix dimension) rotations in the first subdiagonals in parallel. The data transportation assures that each matrix entry is moved into this diagonal exactly once in each sweep. This is only one out of many possibilities.

As as result, after $\frac{n}{2}$ sweeps, the entries are reflected at the main diagonal and the antidiagonal of the matrix with respect to their index. The next sweep brings the entries back to their initial position such that after n sweeps the index positions are restored.

The diagonal of the control matrix $\boldsymbol{N}$ is transported in the same way as the data matrix:

$$\begin{array}{lllll} \mathrm{diag}(& 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 &) & \rightarrow & \mathrm{diag}(\; 2 \;\; 4 \;\; 1 \;\; 6 \;\; 3 \;\; 5 \;) \;\; \rightarrow \\ \mathrm{diag}(& 4 \;\; 6 \;\; 2 \;\; 5 \;\; 1 \;\; 3 &) & \rightarrow & \mathrm{diag}(\; 6 \;\; 5 \;\; 4 \;\; 3 \;\; 2 \;\; 1 \;). \end{array}$$

If the the Jacobi algorithm is able to compute the singular values such that they are ordered as $\boldsymbol{N}$, then after kn ($k \in \mathcal{N}$) sweeps, the large singular values can be read off from the last diagonal entries. They are in ascending order. (Additional sweeps bring them up along the diagonal in descending order.)

3. APPLICATION TO SUBSPACE METHODS

To illustrate the algorithm we consider a spectral estimation problem. The unknown parameters in this case are the sinusoidal frequencies where the received signal $s(i)$ is taken to be the sum of k sinusoids contaminated with an additive zero mean Gaussian white noise $w(i)$:

$$s(i) = \sum_{l=1}^{k} a_l \sin(\omega_l i) + w(i)$$

Taking m samples $s(i)$ $(i = 1, \ldots, m)$ and forming data vectors $\boldsymbol{x}^T(i)$ with dimension n, i.e., $\boldsymbol{x}^T(i) = [s(i)\, s(i-1) \ldots s(i-n+1)]$ the data matrix $\boldsymbol{X}$ is given as follows:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^T(1) \\ \boldsymbol{x}^T(2) \\ \vdots \\ \boldsymbol{x}^T(m) \end{bmatrix}$$

Subspace methods for parameter estimation require the computation of the SVD of the data matrix $\boldsymbol{X}$ in order to determine the signal and the noise subspace. Given the SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k \gg \sigma_{k+1} \geq \ldots \geq \sigma_n$ the signal subspace $\mathcal{S}$ is determined by the right singular vectors associated with the large singular values, i.e., $\mathcal{S} = \text{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ and the noise subspace $\mathcal{N}$ is determined by the right singular vectors associated with the small singular values, i.e., $\mathcal{N} = \text{span}\{\boldsymbol{v}_{k+1}, \ldots, \boldsymbol{v}_n\}$. The gab between the smallest signal singular value σ_k and the largest noise singular value σ_{k+1} is determined by the signal to noise ratio (SNR). Once the signal subspace and the noise subspace are determined the unknown parameters can be computed using ESPRIT[10] or MUSIC.[11]

In our example we use $m = 50$ sample points generated from

$$s(i) = \sin(2\pi 0.1i) + \sin(2\pi 0.25i) + w(i)$$

The data vectors are of dimension $n = 8$. Thereby, data matrices of dimension 30×8 is formed for various values of SNR. In order to determine the signal subspace of $\boldsymbol{X}$ we compute the SVD of $\boldsymbol{X}$ by the original Jacobi SVD method and the sorting Jacobi SVD method. For the sorting Jacobi SVD method $\boldsymbol{N} = \text{diag}(1, 2, \ldots, 8)$ is used. In figure 2 the computed singular values are shown as they appear on the diagonal of $\boldsymbol{\Sigma}$. The ordering of the singular values in $\boldsymbol{\Sigma}$ is the same as the ordering of the singular vectors in $\boldsymbol{V}$. Obviously, if the Jacobi SVD method is used an additional ordering (sorting) of the singular values/vectors is necessary. The position of the small singular values is at random and is influenced by the noise level. A complete sorting is necessary to drop the noise subspace.

If the sorting Jacobi SVD method is used the singular values and singular vectors are obtained in increasing order (as determined by the ordering of $\boldsymbol{N}$). The variation of the singular values in position i depends smoothly on the noise level.

Therefore, the sorting Jacobi method is particularly advantageous in parallel implementations, where the signal subspace is the input for subsequent computations (e.g. in ESPRIT algorithms[10] or subspace model identification[12]). In order to put the different steps of the algorithm together the Jacobi SVD method would require a sorting of the singular values and singular vectors before the data could be transferred to the subsequent computational units. Using the sorting Jacobi SVD method the signal subspace can always be found in the k dominant columns (defined by $\boldsymbol{N}$) of $\boldsymbol{V}$.

4. CONCLUSIONS

In this paper we have presented an algorithm for subspace separation which can be performed on a parallel architecture without sorting the subspace vectors. This was possible by use of an alternative objective function, known in the literature, and corresponding modifications of the processor cells in the triangular array. Performing this sort Jacobi algorithm on a serial architecture does not require explicit introduction of a sorting matrix $\boldsymbol{N}$ because, the order

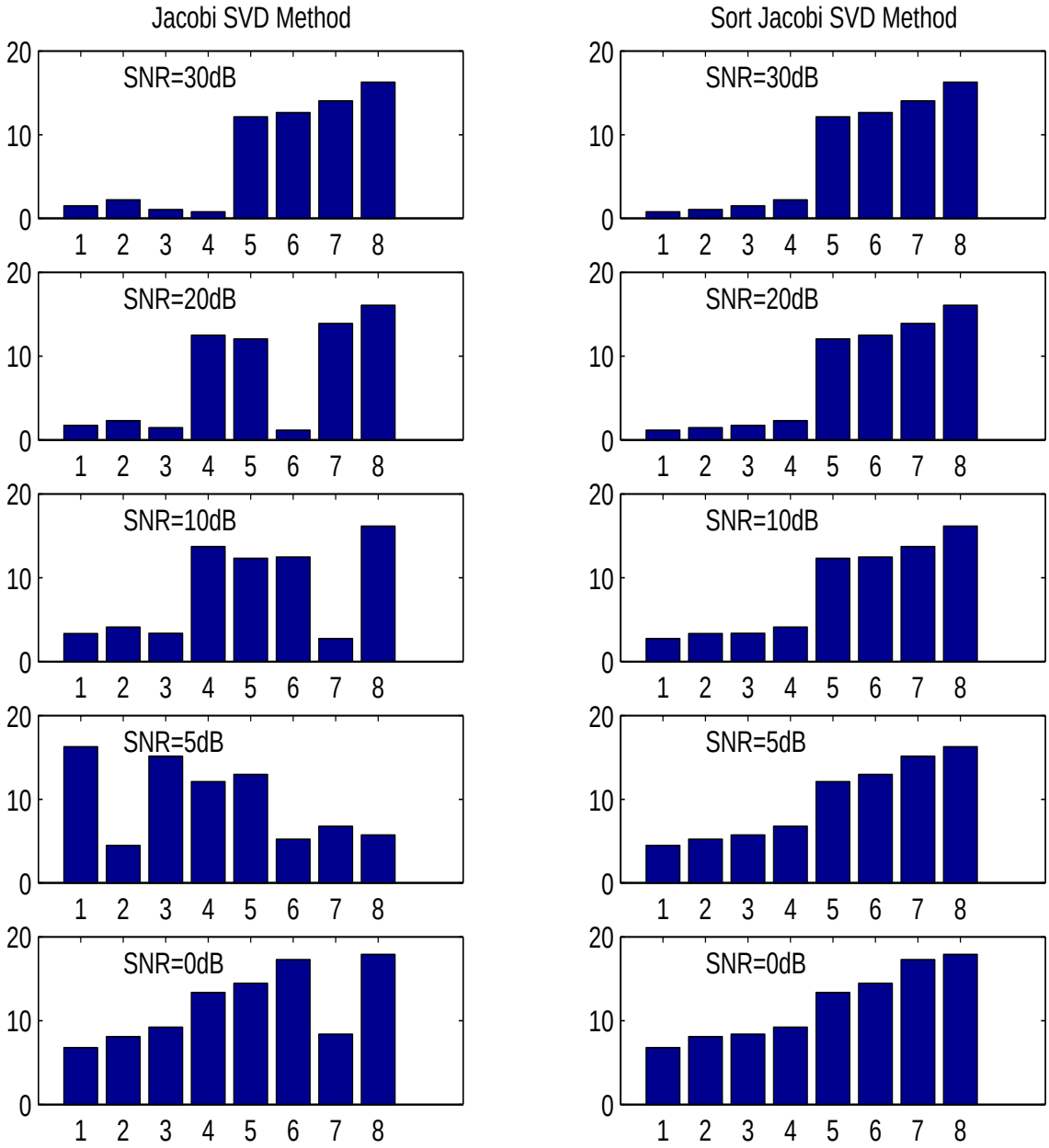

Figure 2. The plots show the values of the singular values (σ_i for $i = 1, \ldots, 8$) as computed by the standard Jacobi SVD method (left) and the sorting Jacobi SVD method (right) for different values of SNR.

relation is fixed for every sweeps. One only has to rotate strictly in the direction such that in each 2×2 sub-problem the diagonal entries are sorted.

On a parallel architecture, the data are transported to obtain optimal parallelism. For this reason, the ordering relation has to be updated after each rotation. The matrix $\boldsymbol{N}$ is also moved around. The cyclic structure of the processing scheme guarantees a strict order relation of the singular values and singular vectors after a certain number of sweeps (if the algorithm is close to convergence).

REFERENCES

1. M. Moonen and B. de Moor, *SVD and Signal Processing III: Algorithms, Applications and Architectures*, Elsevier Science Publishers B. V. (North Holland), Amsterdam, 1995.
2. K. Hüper, J. Götze, and S. Paul, "Subspace separation by discretizations of double brackett flows," in *SVD and Signal Processing III*, pp. 251–258, 1995.
3. U. Helmke and J. B. Moore, *Optimization and dynamical systems*, Springer, London, 1994.
4. R. Brent and F. Luk, "The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Stat. Comput.* **6**, pp. 69–84, 1985.
5. E. Hansen, "On cyclic Jacobi methods," *J. Coc. Indust. Appl. Math.* **11**(2), pp. 448–459, 1963.
6. R. W. Brockett, "Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems," *Lin. Algebra & Applic.* **146**, pp. 79–91, 1991.
7. K. Hüper and U. Helmke, "Structure and convergence of Jacobi-type methods," Tech. Rep. TUM-LNS-TR-95-2, Technical University of Munich, May 1995. submitted to Num. Math.
8. K. Hüper, *Structure and convergence of Jacobi-type methods.* PhD thesis, Technical University Munich, 1996.
9. W. Gentleman and H. Kung, "Matrix triangularization by systolic arrays," in *SPIE Real–Time Signal Processing IV 298*, pp. 19–26, (San Diego (USA)), 1981.
10. R. Roy and T. Kailath, "ESPRIT – estimation of signal parameters via rotational invariance techniques," *IEEE Trans. on Acoust., Speech, Signal Process.* **37**, pp. 984–995, 1989.
11. R. Schmidt, "Multiple emitter location and signal parameter estimation," *IEEE Trans. on Antennas and Propagation* **AP-34**, pp. 276–280, 1986.
12. M. Verhaegen and P. Dewilde. Subspace Model Identifikation. Part I: The output–error state–space model identifikation class of algorithms. *Int. J. Control* 56:1187–1210, 1992.
13. J.R. Cavallaro and F.T. Luk. CORDIC Arithmetic for an SVD Processor. *J. Parallel & Distributed Computing*, 5:271–290, 1988.
14. J.P. Charlier, M. Vanbegin, and P. van Dooren. On Efficient Implementations of Kogbetliantz's Algorithm for Computing the Singular Value Decomposition. *Numer. Math.*, 52:279–300, 1988.
15. J. Demmel and K. Veselic. Jacobi's Method is More Accurate than QR. *SIAM J. Matrix Anal. Appl.*, 13:1204–1245, 1992.
16. M.D. Ercegovac and T. Lang. Redundant and On–Line CORDIC: Application to Matrix Triangularization and SVD. *IEEE Trans. on Computers*, 39:725–740, 1990.
17. G.H. Golub and C.F. van Loan. *Matrix Computations.* The John Hopkins University Press, second edition, 1989.
18. J.G. Proakis and D.G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications.* Macmillan, 2 edition, 1992.
19. G.W. Stewart. A Jacobi–Like Algorithm for Computing the Schur Decomposition of a Nonhermitian Matrix. *SIAM J. Sci. Stat. Comput.*, 6:853–864, 1985