HAL
open science

# Parallel computation of the singular value decomposition

Michael W. Berry, Dany Mezher, Bernard Philippe, Ahmed Sameh

## To cite this version:

Michael W. Berry, Dany Mezher, Bernard Philippe, Ahmed Sameh. Parallel computation of the singular value decomposition. [Research Report] RR-4694, INRIA. 2003. inria-00071892

**HAL Id: inria-00071892**

**https://inria.hal.science/inria-00071892v1**

Submitted on 23 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Parallel computation of the singular value decomposition*

Michael W. Berry , Dani Mezher , Bernard Philippe , Ahmed Sameh

**N°4694**

Janvier 2003

THÈME 4

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4694--FR+ENG

# Parallel computation of the singular value decomposition

Michael W. Berry * , Dani Mezher † , Bernard Philippe ‡ , Ahmed Sameh §

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet ALADIN

Rapport de recherche n ° 4694 — Janvier 2003 — 54 pages

**Abstract:** The goal of this survey is to give a view of the state-of-the-art of computing the Singular Value Decomposition (SVD) of dense and sparse matrices, with some emphasis on those schemes that are suitable for parallel computing platforms. For dense matrices, we survey those schemes that yield the complete decomposition, whereas for sparse matrices we survey schemes that yield only the extremal singular triplets. Special attention is devoted to the computation of the smallest singular values which are normally the most difficult to evaluate but which provide a measure of the distance to singularity of the matrix under consideration. Also, a parallel method for computing pseudospectra, which depends on computing the smallest singular values, is presented at the conclusion of the survey.

**Key-words:** Singular Value Decomposition, SVD, parallel algorithms,Jacobi, subspace iteration, Lanczos, trace minimization, Davidson, Jacobi-Davidson, pseudospectrum

*(Résumé : tsvp)*

* Department of Computer Science, University of Tennessee, berry@cs.utk.edu
† ESIB, Université Saint-Joseph, Beyrouth, dany.mezher@fi.usj.edu.lb
‡ INRIA/IRISA, Rennes, philippe@irisa.fr
§ Department of Computer Science, Purdue University, West Lafayette, sameh@cs.purdue.edu

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

# Calcul parallèle de décompositions aux valeurs singulières

**Résumé :** Nous tentons dans ce rapport un état de l'art pour le calcul parallèle de décompositions aux valeurs singulières de matrices. Dans le cas de matrices pleines, la décomposition est complète tandis que dans le cas des matrices creuses, seules les valeurs extrémales sont visées. Nous considérons en particulier le calcul des plus petites valeurs singulières qui sont les plus difficiles à estimer mais qui permettent de mesurer la distance à la singularité de la matrice. Une méthode parallèle pour déterminer un pseudospectre est présentée en guise de conclusion.

**Mots-clé :** Décomposition aux valeurs singulières, algorithme parallèle, Jacobi, itération simultannée, Lanczos, minimisation de la trace, Davidson, Jacobi-Davidson, pseudospectre

# Contents

**1 Introduction** **4**
1.1 Basics . . . 4
1.2 Sensitivity of the smallest singular value . . . 6
1.3 Distance to singularity - Pseudospectrum . . . 7

**2 Jacobi Methods for Dense Matrices** **10**
2.1 Two-sided Jacobi scheme [2JAC] . . . 10
2.2 One-sided Jacobi scheme [1JAC] . . . 13
2.3 Algorithm [QJAC] . . . 16

**3 Methods for large and sparse matrices** **19**
3.1 Sparse storages and linear systems . . . 19
3.2 Subspace iteration [SISVD] . . . 20
3.3 Lanczos methods . . . 22
3.3.1 The Single-vector Lanczos method [LASVD] . . . 22
3.3.2 The Block Lanczos method [BLSVD] . . . 24
3.4 The Trace Minimization method [TRSVD] . . . 27
3.4.1 Polynomial acceleration techniques for [TRSVD] . . . 29
3.4.2 Shifting strategy for [TRSVD] . . . 29
3.5 Refinement of Left Singular Vectors . . . 31
3.6 The Davidson methods . . . 34
3.6.1 General framework of the methods . . . 34
3.6.2 How do the Davidson methods differ ? . . . 36
3.6.3 Application to the computation of the smallest singular value . . . 38

**4 Parallel computation for sparse matrices** **40**
4.1 Parallel sparse matrix-vector multiplications . . . 40
4.2 A parallel scheme for basis orthogonalization . . . 41
4.3 Computing the smallest singular value on several processors . . . 44

**5 Application : parallel computation of a pseudo-spectrum** **45**
5.1 Parallel Path Following Algorithm using Triangles . . . 45
5.2 Speedup and Efficiency . . . 47
5.3 Test problems . . . 47

# 1 Introduction

## 1.1 Basics

The Singular Value Decomposition (SVD) is a powerful computational tool. Modern algorithms for obtaining such a decomposition of general matrices have had a profound impact on numerous applications in science and engineering disciplines. In what follows, we will provide a brief survey of some parallel algorithms for obtaining the singular value decomposition for dense, and large sparse matrices. For sparse matrices, however, we focus mainly on the problem of obtaining the smallest singular triplets.

In order to introduce the notations of the chapter, the basic facts related to SVD are presented without proof. Complete presentations are given in many text books, as for instance [42, 88].

**Theorem 1.1 (SVD)** *If $A \in \mathbb{R}^{m\times n}$ is a real matrix, then there exist orthogonal matrices*

$$U = [u_1, \cdots, u_m] \in \mathbb{R}^{m\times m} \text{ and } V = [v_1, \cdots, v_n] \in \mathbb{R}^{n\times n}$$

*such that*

$$\Sigma = U^T AV = \mathrm{diag}(\sigma_1, \cdots, \sigma_p) \in \mathbb{R}^{m\times n}, \quad p = \min(m, n) \tag{1}$$

*where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.*

**Definition 1.1** *The singular values of $A$ are the real numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$. They are uniquely defined. For every singular value $\sigma_i$ $(i = 1, \ldots, p)$, the vectors $u_i$ and $v_i$ are respectively the left and right singular vectors of $A$ associated with $\sigma_i$.*

More generally, the theorem holds in the complex field but with the inner products and orthogonal matrices replaced by the hermitian products and unitary matrices, respectively. The singular values, however, remain real non-negative values.

**Theorem 1.2** *Let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ have the singular value decomposition*

$$U^T AV = \Sigma.$$

*Then, the symmetric matrix*

$$C = A^T A \in \mathbb{R}^{n\times n}, \tag{2}$$

*has eigenvalues ${\sigma_1}^2 \geq \cdots \geq {\sigma_n}^2 \geq 0$, corresponding to the eigenvectors $(v_i)$, $(i = 1, \cdots, n)$.*

*The symmetric matrix*

$$B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \tag{3}$$

*has eigenvalues* $\pm\sigma_1, \ldots, \pm\sigma_n$, *corresponding to the eigenvectors*

$$\frac{1}{\sqrt{2}} \begin{pmatrix} u_i \\ \pm v_i \end{pmatrix}, \quad i = 1, \ldots, n.$$

*The matrix* $B$ *is called the augmented matrix.*

Every method for computing singular values is based on one of these two matrices.

The numerical accuracy of the $i$th approximate singular triplet $(\tilde{u}_i, \tilde{\sigma}_i, \tilde{v}_i)$ as determined via the eigensystem of the $2-cyclic$ matrix $B$ is then determined by the norm of the eigenpair residual vector $r_i$ defined by

$$\| r_i \|_2 = [\| B(\tilde{u}_i, \tilde{v}_i)^T - \tilde{\sigma}_i(\tilde{u}_i, \tilde{v}_i)^T \|_2] / [\| \tilde{u}_i \|_2^2 + \| \tilde{v}_i \|_2^2]^{\frac{1}{2}},$$

which can also be written as

$$\| r_i \|_2 = [(\| A\tilde{v}_i - \tilde{\sigma}_i\tilde{u}_i \|_2^2 + \| A^T\tilde{u}_i - \tilde{\sigma}_i\tilde{v}_i \|_2^2)^{1/2}] / \| \tilde{u}_i \|_2^2 + \| \tilde{v}_i \|_2^2]^{1/2} \tag{4}$$

Alternatively, we may compute the SVD of $A$ indirectly by the eigenpairs of either the $n \times n$ matrix $A^TA$ or the $m \times m$ matrix $AA^T$. If $V = \{v_1, v_2, \ldots, v_n\}$ is the $n \times n$ orthogonal matrix representing the eigenvectors of $A^TA$, then

$$V^T(A^TA)V = diag(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, \underbrace{0, \ldots, 0}_{n-r}),$$

where $\sigma_i$ is the $i$th nonzero singular value of $A$ corresponding to the right singular vector $v_i$. The corresponding left singular vector, $u_i$, is then obtained as $u_i = \frac{1}{\sigma_i}Av_i$. Similarly, if $U = \{u_1, u_2, \ldots, u_m\}$ is the $m \times m$ orthogonal matrix representing the eigenvectors of $AA^T$, then

$$U^T(AA^T)U = diag(\sigma_1^2, \sigma_2^2, \ldots, \sigma_r^2, \underbrace{0, \ldots 0}_{m-r}),$$

where $\sigma_i$ is the $i$th nonzero singular value of $A$ corresponding to the left singular vector $u_i$. The corresponding right singular vector, $v_i$, is then obtained as $v_i = \frac{1}{\sigma_i}A^Tu_i$.

Computing the SVD of $A$ via the eigensystems of either $A^TA$ or $AA^T$ may be adequate for determining the largest singular triplets of $A$, but some loss of accuracy may be observed for the smallest singular triplets (see [20]). In fact, extremely small singular values of $A$ may be computed as zero eigenvalues of $A^TA$ (or $AA^T$). Whereas the smallest and largest singular values of $A$ are the extremes of the spectrum of $A^TA$ or $AA^T$, the smallest singular values of $A$ lie at the center of the spectrum of $B$ in (3). For computed eigenpairs of $A^TA$ and $AA^T$, the norms of the $i$th eigenpair residuals (corresponding to (4)) are given by

$$\| r_i \|_2 = \| A^TA\tilde{v}_i - \tilde{\sigma}_i^2\tilde{v}_i \|_2 \,/\, \| \tilde{v}_i \|_2 \quad \text{and} \quad \| r_i \|_2 = \| AA^T\tilde{u}_i - \tilde{\sigma}_i^2\tilde{u}_i \|_2 \,/\, \| \tilde{u}_i \|_2,$$

respectively. Thus, extremely high precision in computed eigenpairs may be necessary to compute the smallest singular triplets of $A$. This fact is analyzed in Section 1.2.

When $A$ is a square nonsingular matrix, it may be advantageous in certain cases to compute the singular values of $A^{-1}$ which are $\frac{1}{\sigma_n} \geq \cdots \geq \frac{1}{\sigma_1}$. This approach has the drawback of solving linear systems involving

the matrix $A$, but when manageable, it provides a more robust algorithm. Such an alternative is of interest for some subspace methods (see Section 3). Actually, the method can be extended to rectangular matrices of full rank by considering a QR-factorization:

**Proposition 1.1** *Let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ be of rank $n$. Let*

$$A = QR, \text{ where } Q \in \mathbb{R}^{m\times n} \text{ and } R \in \mathbb{R}^{n\times n},$$
$$\text{such that } Q^TQ = I_n \text{ and } R \text{ is upper triangular.}$$

*The singular values of $R$ are the same as the singular values of $A$.*

Therefore, the smallest singular value of $A$ can be computed from the largest eigenvalue of $(R^{-1}R^{-T})$ or of $\begin{pmatrix} 0 & R^{-1} \\ R^{-T} & 0 \end{pmatrix}$.

## 1.2 Sensitivity of the smallest singular value

In order to compute the smallest singular value in a reliable way, one must investigate the sensitivity of the singular values with respect to perturbations of the matrix at hand.

**Theorem 1.3** *Let $A \in \mathbb{R}^{n\times n}$ and $\Delta \in \mathbb{R}^{n\times n}$. The singular values of $A$ and $A + \Delta$ are respectively denoted*

$$\begin{aligned} \sigma_1 \geq \sigma_2 &\geq \cdots \geq \sigma_n \\ \tilde{\sigma}_1 \geq \tilde{\sigma}_2 &\geq \cdots \geq \tilde{\sigma}_n. \end{aligned}$$

*They satisfy the following bounds*

$$|\sigma_i - \tilde{\sigma}_i| \leq \|\Delta\|_2, \text{ for } i = 1, \cdots, n.$$

**Proof.** See [88]. ◇

When applied to the smallest singular value, this result ends up with the following estimation.

**Proposition 1.2** *The condition number of the smallest singular value of $A$ is equal to $\chi_2(A) = \|A\|_2\|A^{-1}\|_2$.*

**Proof.** The result is obtained from

$$\frac{|\sigma_n - \tilde{\sigma}_n|}{\sigma_n} \leq \left(\frac{\|\Delta\|_2}{\|A\|_2}\right)\frac{\|A\|_2}{\sigma_n}.$$

◇

This means that the smallest singular value of an ill-conditioned matrix cannot be computed with high accuracy even with an algorithm of perfect arithmetic behaviour (ie., backward stable).

Recently, some progress has been made. In [24] it is shown that for some special class of matrices, an accurate computation of the smallest singular value may be obtained via a combination of some QR factorization with column pivoting and a one-sided Jacobi algorithm (see Section 2.2).

Since the singular values are roots of a polynomial (e.g. roots of the characteristic polynomial of the augmented matrix), then when simple, they are differentiable with respect to the entries of the matrix. More precisely, one can states that :

**Theorem 1.4** *Let $\sigma$ be a simple singular value of the matrix $A = (a_{ij})$ with $u = (u_i)$ and $v = (v_i)$ being the corresponding normalized left and right singular vectors. Then, the singular value is differentiable with respect to the matrix $A$, or*

$$\frac{\partial \sigma}{\partial a_{ij}} = u_i v_j, \ \forall i, \ j = 1, \ldots, n.$$

**Proof.** See [89]. ◇

## 1.3 Distance to singularity - Pseudospectrum

Let us consider a linear system defined by the square matrix $A \in \mathbb{R}^{n \times n}$. Often, one needs to quantify how far is the system under consideration from being singular. It turns out that the smallest singular value $\sigma_{\min}(A)$ is equal to that distance.

Let $\mathcal{S}$ be the set of all singular matrices in $\mathbb{R}^{n \times n}$ and the distance corresponding to the 2-norm : $d(A, B) = \|A - B\|_2$ for $A, \ B \in \mathbb{R}^{n \times n}$.

**Theorem 1.5** $d(A,\mathcal{S}) = \sigma_{\min}(A)$.

**Proof.** Let us denote $\sigma = \sigma_{\min}(A)$ and $d = d(A, \mathcal{S})$. There exist two unitary vectors $u$ and $v$ such that $Au = \sigma v$. Therefore $(A - \sigma v u^T)u = 0$. Since $\|\sigma v u^T\|_2 = \sigma$, it appears that $B = A - \sigma v u^T \in \mathcal{S}$ and $d(A, B) = \sigma$ which proves that $d \leq \sigma$.

Conversely, let us consider any matrix $\Delta \in \mathbb{R}^{n \times n}$ such that $(A + \Delta) \in \mathcal{S}$. There exists a unitary vector $u$ such that $(A + \Delta)u = 0$. Therefore :

$$\sigma \leq \|Au\|_2 = \|\Delta u\|_2 \leq \|\Delta\|_2,$$

which concludes the proof. ◇

This result leads to a useful lower bound of the condition number of a matrix.

**Proposition 1.3** *The condition number $\chi_2(A) = \|A\|_2 \|A^{-1}\|_2$ satisfies*

$$\chi_2(A) \geq \frac{\|A\|_2}{\|A - B\|_2},$$

*for any singular matrix $B \in \mathcal{S}$.*

**Proof.** The result follows from the fact that $B = A + (B - A)$ and $\|A^{-1}\|_2 = \frac{1}{\sigma_{\min}(A)}$. ◇

For instance in [49], this property is used to illustrate that the condition number of the linear systems arising from the simulation of flow in porous media, using mixed finite element methods, is of the order of the ratio of the extreme values of conductivity.

Let us now consider the sensitivity of eigenvalues with respect to matrix perturbations. Towards this goal, the notion of pseudospectrum [91] or of $\epsilon$-spectrum [36] was introduced :

**Definition 1.2** *For a matrix $A \in \mathbb{R}^{n\times n}$ (or $A \in \mathbb{C}^{n\times n}$) and a parameter $\epsilon > 0$, the pseudospectrum is the set :*

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid \exists \Delta \in \mathbb{C}^{n\times n} \text{ such that } \|\Delta\| \leq \epsilon \text{ and } z \text{ is an eigenvalue of } (A + \Delta)\}. \tag{5}$$

This definition does not provide a constructive method for determining the pseudospectrum. Fortunately, a constructive method can be drawn from the following property.

**Proposition 1.4** *The pseudspectrum is the set :*

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid \sigma_{\min}(A - zI) \leq \epsilon\}, \tag{6}$$

*where $I$ is the identity matrix of order $n$.*

**Proof.** For any $z \in \mathbb{C}$, $z$ is an eigenvalue of $(A + \Delta)$ if and only if the matrix $(A - zI) + \Delta$ is singular. The proposition is therefore a straight application of Theorem 1.5. ◇

This proposition provides a criterion for deciding whether $z$ belongs to $\Lambda_\epsilon(A)$. To represent the pseudospectrum graphically, one can define a grid in the complex region under concideration and compute $\sigma_{\min}(A - z_{ij}I)$, for all the $z_{ij}$ determined by the grid. Although highly parallel, this approach involves a very high volume of operations. Presently, one prefers to use path following techniques [15, 8, 61]. Section 5 describes one of these methods. For illustration, the pseudospectrum of a matrix from the Matrix Market [59] test suite is displayed in Figure 1, where several values of $\epsilon$ are shown.

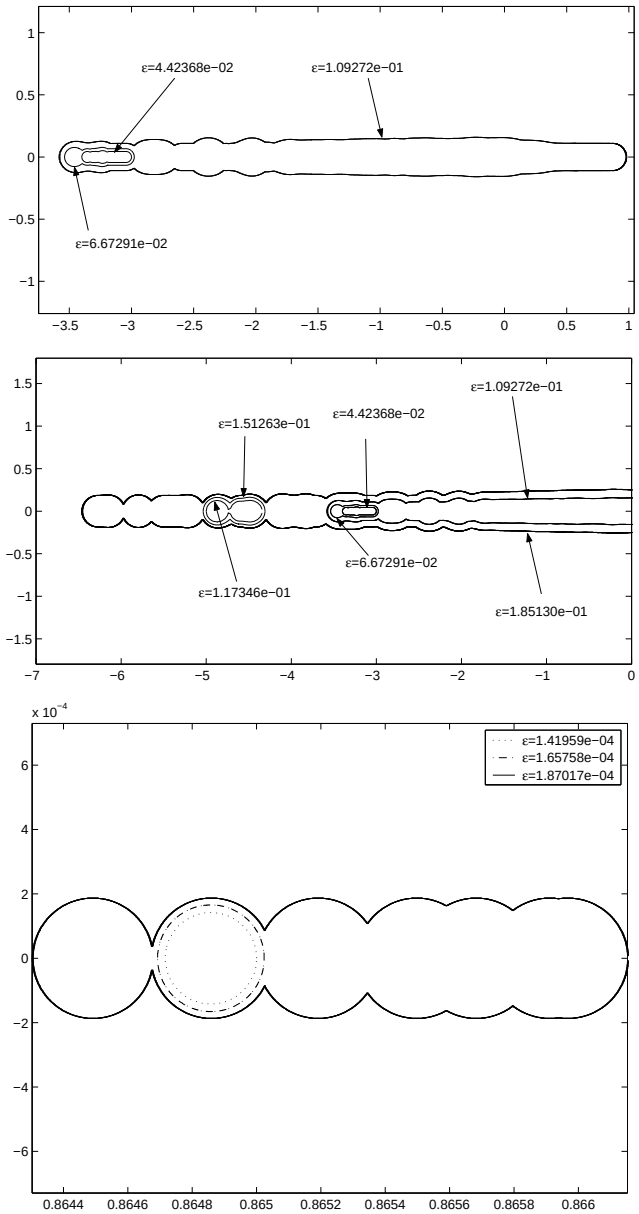



Figure 1: Portraits of the matrix DW8192.

# 2 Jacobi Methods for Dense Matrices

## 2.1 Two-sided Jacobi scheme [2JAC]

Consider the standard eigenvalue problem

$$Ax = \lambda x \tag{7}$$

where $A$ is a real $n \times n$-dense symmetric matrix. One of the best unknown methods for determining all the eigenpairs of (7) was developed by the nineteenth century mathematician, Jacobi. We recall that Jacobi's sequential method reduces the matrix $A$ to the diagonal form by an infinite sequence of plane rotations

$$A_{k+1} = U_k A_k U_k^T, \quad k = 1, 2, ...,$$

where $A_1 \equiv A$, and $U_k = U_k(i, j, \theta_{ij}^k)$ is a rotation of the $(i, j)$-plane where

$$u_{ii}^k = u_{jj}^k = c_k = \cos\theta_{ij}^k \quad \text{and} \quad u_{ij}^k = -u_{ji}^k = s_k = \sin\theta_{ij}^k.$$

The angle $\theta_{ij}^k$ is determined so that $a_{ij}^{k+1} = a_{ji}^{k+1} = 0$, or

$$\tan 2\theta_{ij}^k = \frac{2a_{ij}^k}{a_{ii}^k - a_{jj}^k},$$

where $|\theta_{ij}^k| \leq \frac{1}{4}\pi$.

For numerical stability, we determine the plane rotation by

$$c_k = \frac{1}{\sqrt{1 + t_k^2}} \quad \text{and} \quad s_k = c_k t_k,$$

where $t_k$ is the smaller root (in magnitude) of the quadratic equation

$$t_k^2 + 2\alpha_k t_k - 1 = 0, \quad \alpha_k = \cot 2\theta_{ij}^k.$$

Hence, $t_k$ may be written as

$$t_k = \frac{\text{sign } \alpha_k}{|\alpha_k| + \sqrt{1 + \alpha_k^2}}.$$

Each $A_{k+1}$ remains symmetric and differs from $A_k$ only in rows and columns $i$ and $j$, where the modified elements are given by

$$\begin{aligned} a_{ii}^{k+1} &= a_{ii}^k + t_k a_{ij}^k, \\ a_{jj}^{k+1} &= a_{jj}^k - t_k a_{ij}^k, \end{aligned}$$

and

$$a_{ir}^{k+1} = c_k a_{ir}^k + s_k a_{jr}^k, \tag{8}$$

$$a_{jr}^{k+1} = -s_k a_{ir}^k + c_k a_{jr}^k, \tag{9}$$

in which $r \neq i, j$. If we represent $A_k$ by

$$A_k = D_k + E_k + E_k^T, \tag{10}$$

where $D_k$ is diagonal and $E_k$ is strictly upper triangular, then as $k$ increases $\| E_k \|_F$ approaches zero, and $A_k$ approaches the diagonal matrix $\Lambda = \text{diag}\ (\lambda_1, \lambda_2, \ldots, \lambda_n)$ ($\| \cdot \|_F$ denotes the Frobenius norm). Similarly, the transpose of the product $(U_k \cdots U_2 U_1)$ approaches a matrix whose $j$th column is the eigenvector corresponding to $\lambda_j$.

Several schemes are possible for selecting the sequence of elements $a_{ij}^k$ to be eliminated via the plane rotations $U_k$. Unfortunately, Jacobi's original scheme, which consists of sequentially searching for the largest off-diagonal element, is too time consuming for implementation on a multiprocessor. Instead, a simpler scheme in which the off-diagonal elements $(i, j)$ are annihilated in the cycle fashion $(1,2), (1,3), \ldots, (1,n)$, $(2,3), \ldots, (2,n), \ldots, (n-1,n)$ is usually adopted as its convergence is assured [31]. We refer to each sequence of $n$ rotations as a sweep. Furthermore, quadratic convergence for this sequential cyclic Jacobi scheme has been well documented (see [82, 95]). Convergence usually occurs within 6 to 10 sweeps, i.e., from $3n^2$ to $5n^2$ Jacobi rotations.

A parallel version of this cyclic Jacobi algorithm is obtained by the simultaneous annihilation of several off-diagonal elements by a given $U_k$, rather than only one as is done in the serial version. For example, let $A$ be of order 8 and consider the orthogonal matrix $U_k$ as the direct sum of 4 independent plane rotations, where the $c_i$'s and $s_i$'s for $i = 1, 2, 3, 4$ are simultaneously determined. An example of such a matrix is

$$R_k(1,3) \oplus R_k(2,4) \oplus R_k(5,7) \oplus R_k(6,8),$$

where $R_k(i,j)$ is that rotation which annihilates the $(i,j)$ off-diagonal element. If we consider one sweep to be a collection of orthogonal similarity transformations that annihilate the element in each of the $\frac{1}{2}n(n-1)$ off-diagonal positions (above the main diagonal) only once, then for a matrix of order 8, the first sweep will consist of 8 successive orthogonal transformations with each one annihilating distinct groups of 4 elements simultaneously. For the remaining sweeps, the structure of each subsequent transformation $U_k, k > 8$, is chosen to be the same as that of $U_j$ where $j = 1 + (k-1) \bmod 8$. In general, the most efficient annihilation scheme consists of $(2r-1)$ similarity transformations per sweep, where $r = \lfloor \frac{1}{2}(n+1) \rfloor$, in which each transformation annihilates different $\lfloor \frac{1}{2}n \rfloor$ off-diagonal elements (see [78]). Although several annihilation schemes are possible, the Jacobi algorithm we present below utilizes an annihilation scheme which requires a minimal amount of indexing for computer implementation. Moreover Luk and Park [57, 56] have demonstrated that various parallel Jacobi rotation ordering schemes are equivalent to the sequential row ordering scheme, and hence share the same convergence properties.

## Algorithm [2JAC]

*STEP* 1: (Apply orthogonal similarity transformations via $U_k$ for current sweep).

1. (a) For $k = 1, 2, 3, \ldots, n-1$ (serial loop)

simultaneously annihilate elements in position $(i, j)$, where

$$\begin{cases} i = 1, 2, 3, \ldots, \lceil \frac{1}{2}(n-k) \rceil , \\ j = (n-k+2) - i. \end{cases}$$

and additionally for $k > 2$,

$$\begin{cases} i = (n-k+2), (n-k+3), \ldots, n - \lfloor \frac{1}{2}k \rfloor , \\ j = (2n-k+2) - i. \end{cases}$$

(b) For $k = n$ simultaneously annihilate elements in positions $(i, j)$, where

$$\begin{cases} i = 2, 3, \ldots, \quad \lceil \frac{1}{2}n \rceil \\ j = (n+2) - i \end{cases}$$

*STEP 2*: (Convergence test).

2. (a) Compute $\| D_k \|_F$ and $\| E_k \|_F$ (see (10)).

   (b) If

$$\frac{\| E_k \|_F}{\| D_k \|_F} < tolerance, \tag{11}$$

   then stop.

   Otherwise, go to Step 1 to begin next sweep.

We note that this algorithm requires $n$ similarity transformations per sweep for a dense real symmetric matrix of order $n$ ($n$ may be even or odd). Each $U_k$ is the direct sum of either $\lfloor \frac{1}{2}n \rfloor$ or $\lfloor \frac{1}{2}(n-1) \rfloor$ plane rotations, depending on whether $k$ is odd or even, respectively. The annihilation pattern for $n = 8$ is shown in table 1, where the integer $k$ denotes an element annihilation via $U_k$.

In the annihilation of a particular $(i, j)$-element in Step 1 above, we update the off-diagonal elements in rows and columns $i$ and $j$ as specified by (8) and (9). With regard to storage requirements, it would be advantageous to modify only those row or column entries above the main diagonal and utilize the guaranteed symmetry of $A_k$. However, if one wishes to take advantage of the vectorization that may be supported by the parallel computing platform, we disregard the symmetry of $A_k$ and operate with full vectors on the entirety of rows and columns $i$ and $j$ in (8) and (9), i.e., we are using a full matrix scheme. The product of the $U_k$'s, which eventually yields the eigenvectors for $A$, is accumulated in a separate two-dimensional array by applying (8) and (9) to the $n \times n$-identity matrix.

In Step 2, we monitor the convergence of the algorithm by using the ratio of the computed norms to measure the systematic decrease in the relative magnitudes of the off-diagonal elements with respect to the relative magnitudes of the diagonal elements. For double precision accuracy in the eigenvalues and eigenvectors, a *tolerance* of order $10^{-16}$ will suffice for Step 2(b). If we assume convergence (see [56]), this multiprocessor algorithm can be shown to converge quadratically by following the work of Henrici [45] and Wilkinson [95].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| x | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | x | 5 | 4 | 3 | 2 | 1 | 8 |
| | | x | 3 | 2 | 1 | 8 | 7 |
| | | | x | 1 | 8 | 7 | 6 |
| | | | | x | 7 | 6 | 5 |
| | | | | | x | 5 | 4 |
| | | | | | | x | 3 |
| | | | | | | | x |

Table 1: Annihilation scheme for [2JAC].

## 2.2 One-sided Jacobi scheme [1JAC]

If $A$ in (7) is not only symmetric but also definite, it is considerably more efficient to apply a *one-sided* Jacobi method which in effect only post-multiplies $A$ by plane rotations. The derivation of the *one-sided* Jacobi method is motivated by the singular value decomposition of rectangular matrices. Specifically, suppose $A$ is a real $m \times n$-matrix with $m \gg n$ and rank $A = r$. The singular value decomposition of $A$ can be defined as

$$A = U\Sigma V^T, \tag{12}$$

where $U^TU = V^TV = I_n$ and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$, $\sigma_i > 0$ for $1 \le i \le r$, $\sigma_j = 0$ for $j \ge r+1$. The first $r$ columns of the orthonormal matrix $U$ and the orthogonal matrix $V$ define the orthonormalized eigenvectors associated with the $r$ nonzero eigenvalues of $AA^T$ or $A^TA$.

As indicated in [79] for a ring of processors, using a method based on the one-sided iterative orthogonalization method of Hestenes (see also [51, 64]) is an efficient way to compute the decomposition (12). Luk [58] recommended this singular value decomposition scheme on the Illiac IV, and corresponding systolic algorithms associated with *two-sided* schemes have been presented in [13] and [14]. We now consider a few modifications to the scheme discussed in [79] for the determination of (12) on shared memory multiprocessors, e.g., see [11].

Our main goal is to determine the orthogonal matrix $V = [\tilde{V}, \tilde{W}]$, where $\tilde{V}$ is $n \times r$, so that

$$A\tilde{V} = Q = (q_1, q_2, \ldots, q_r), \tag{13}$$

and

$$q_i^T q_j = \sigma_i^2 \delta_{ij},$$

where the columns of $A$ are orthogonal and $\delta_{ij}$ is the Kronecker-delta. Writing $Q$ as

$$Q = \tilde{U}\tilde{\Sigma} \quad \text{with} \quad \tilde{U}^T\tilde{U} = I_r, \quad \text{and} \quad \tilde{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_r),$$

then

$$A = \tilde{U}\tilde{\Sigma}\tilde{V}^T.$$

We construct the matrix $V$ via the plane rotations

$$(a_i, a_j) \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = (\tilde{a}_i, \tilde{a}_j), \quad i < j,$$

so that

$$\tilde{a}_i^T \tilde{a}_j = 0 \quad \text{and} \quad \| \tilde{a}_i \|_2 > \| \tilde{a}_j \|_2, \tag{14}$$

where $a_i$ designates the $i$th column of matrix $A$. This is accomplished by choosing

$$c = \left[ \frac{\beta + \gamma}{2\gamma} \right]^{1/2} \quad \text{and} \quad s = \left[ \frac{\alpha}{2\gamma c} \right] \quad \text{if} \quad \beta > 0, \tag{15}$$

or

$$s = \left[ \frac{\gamma - \beta}{2\gamma} \right]^{1/2} \quad \text{and} \quad c = \left[ \frac{\alpha}{2\gamma s} \right] \quad \text{if} \quad \beta < 0, \tag{16}$$

where $\alpha = 2a_i^T a_j$, $\beta = \| a_j \|_2^2$, and $\gamma = (\alpha^2 + \beta^2)^{1/2}$. Note that (14) requires the columns of $Q$ to decrease in norm from left to right, and hence the resulting $\sigma_i$ to be in monotonic nonincreasing order. Several schemes can be used to select the order of the $(i, j)$-plane rotations. Following the annihilation pattern of the off-diagonal elements in the sequential Jacobi algorithm mentioned in Section 1, we could certainly orthogonalize the columns in the same cyclic fashion and thus perform the one-sided orthogonalization serially. This process is iterative with each sweep consisting of $\frac{1}{2}n(n-1)$ plane rotations selected in cyclic fashion.

By adopting the ordering of the annihilation scheme in [2JAC], we obtain a parallel version of the one-sided Jacobi method for computing the singular value decomposition on a multiprocessor. For example, let $n = 8$ and $m \gg n$ so that in each sweep of our one-sided Jacobi algorithm we simultaneously orthogonalize pairs of columns of A (see Table 1). For example, for $n = 8$ we can orthogonalized the pairs (1,8), (2,7), (3,6), (4,5) simultaneously via postmultiplication by a matrix $V_i$ which consists of the direct sum of 4 plane rotations. In general, each $V_k$ will have the same form as $U_k$ so that at the end of any particular sweep $s_i$ we have

$$V_{s_1} = V_1 V_2 \cdots V_n,$$

and hence

$$V = V_{s_1} V_{s_2} \cdots V_{s_t}, \tag{17}$$

where $t$ is the number of sweeps required for convergence.

## Algorithm [1JAC]

*STEP* 1: (Postmultiply matrix $A$ by orthogonal matrix $V_k$ for current sweep).

1. (a) Initialize the convergence counter, *istop*, to zero.

(b) For $k = 1, 2, 3, \ldots, n-1$ (serial loop)
simultaneously orthogonalize the column pairs $(i, j)$, where $i$ and $j$ are given by 1(a) in Step 1 of [2JAC], provided that for each $(i, j)$ we have

$$\frac{(a_i^T a_j)^2}{(a_i^T a_i)(a_j^T a_j)} > tolerance, \tag{18}$$

and $i$, $j \in \{k | k < k_{\min}\}$, where $k_{\min}$ is the minimal column index $k$ such that $\| a_k \|_2^2 < tolerance$. Upon the termination of [1JAC], $r = \text{rank } A = k_{\min}$. Note: if (18) is not satisfied for any particular pair $(i, j)$, *istop* is incremented by 1 and that rotation is not performed.

(c) For $k = n$
simultaneously orthogonalized the column pairs $(i, j)$, where $i$ and $j$ are given by 1(b) in Step 1 of [2JAC].

*STEP* 2: (Convergence test).
If *istop* $= \frac{1}{2}n(n-1)$, then compute $\sigma_i = \sqrt{(A^T A)_{ii}}$, $i = 1, 2, \ldots, k_{\min} = r$, and stop. Otherwise, go to beginning of Step 1 to start next sweep.

In the orthogonalization of columns in Step 1, we are implementing the plane rotations specified by (15) and (16), and hence guaranteeing the ordering of column norms and singular values upon termination. Whereas [2JAC] must update rows and columns following each similarity transformation, [1JAC] performs only postmultiplication of $A$ by each $V_k$ and hence the plane rotation $(i, j)$ changes only the elements in columns $i$ and $j$ of matrix $A$. The changed elements can be represented by

$$a_i^{k+1} = ca_i^k + sa_j^k, \tag{19}$$

$$a_j^{k+1} = -sa_i^k + ca_j^k, \tag{20}$$

where $a_i$ denotes the $i$th column of matrix $A$, and $c$, $s$ are determined by either (15) or (16). Since no row accesses are required and no columns are interchanged, one would expect good performance for this method on a machine which can apply vector operations to compute (19) and (20). Each processor is assigned one rotation and hence orthogonalizes one pair of the $n$ columns of matrix $A$.

Following the convergence test used in [64], in Step 2, we count the number of times the quantity

$$\frac{a_i^T a_j}{(a_i^T a_i)(a_j^T a_j)} \tag{21}$$

falls, in any sweep, below a given *tolerance*. The algorithm terminates when the counter *istop* reaches $\frac{1}{2}n(n-1)$, the total number of column pairs, after any sweep. Upon termination, the first $r$ columns of the matrix $A$ are overwritten by the matrix $Q$ from (13) and hence the nonzero singular values $\sigma_i$ can be obtained via the $r$ square roots of the first $r$ diagonal entries of $A^T A$. The matrix $\tilde{U}$ in (12), which contains the leading $r$, left singular vectors of the original matrix $A$, is readily obtained by column scaling of the resulting matrix $A$ (now overwritten by $Q = \tilde{U}\tilde{\Sigma}$) by the nonzero singular values $\sigma_i$. Similarly, the matrix $V$, which contains the right singular vectors of the original matrix $A$, is obtained as in (17) as the product

of the orthogonal $V_k$'s. This product is accumulated in a separate two-dimensional array by applying the rotations specified by (19) and (20) to the $n \times n$-identity matrix. It is important to note that the use of the ratio in (21) is preferable over the use of $a_i^T a_j$, since this dot-product is necessarily small for relatively small singular values.

Although our derivation of [1JAC] concerned the singular value decomposition of rectangular matrices, it is most effective for solving the eigenvalue problem in (7) for symmetric positive definite matrices. If $m = n$, $A$ is a positive definite matrix, and $Q$ (13) is an orthogonal matrix. Consequently, it is not difficult to show that

$$\begin{cases} \sigma_i = \lambda_i \\ x_i = \frac{q_i}{\lambda_i}, \end{cases} \quad i = 1, 2, \ldots, n,$$

where $\lambda_i$ denotes the $i$th eigenvalue of $A$, $x_i$ the corresponding normalized eigenvector, and $q_i$ the $i$th column of matrix $Q$. If $A$ is symmetric, but perhaps not positive definite, we can obtain its eigenvectors by considering instead $A + \alpha I$, where $\alpha$ is the smallest quantity that ensures that definiteness of $A + \alpha I$, and retrieve the eigenvalues of $A$ via Rayleigh quotients.

[1JAC] has two advantages over [2JAC]: (i) no row accesses are needed, and (ii) the matrix $\mathcal{Q}$ need not be accumulated.

## 2.3 Algorithm [QJAC]

As discussed above, [1JAC] is certainly a viable candidate for computing the singular value decomposition in (12) on multiprocessors . However, for $m \times n$-matrices $A$ in which $m \gg n$, the problem complexity can be reduced if an initial orthogonal factorization of $A$ is performed. One can then apply the *one-sided* Jacobi method, [1JAC], to the resulting upper-triangular matrix $R$ (which may be singular) and obtain the decomposition (12). In this section we present a multiprocessor method, QJAC, which can be quite effective for computing (12) on parallel machines.

Given the $m \times n$-matrix $A$, where $m \gg n$, we perform a block generalization of Householder's reduction for the orthogonal factorization

$$A = QR, \tag{22}$$

where $Q$ is $m \times n$-orthonormal matrix, and $R$ is an $n \times n$-upper-triangular matrix. The block schemes of LAPACK are used for computing (22) to make use of vector-matrix, matrix-vector (BLAS2), and matrix-matrix (BLAS3) multiplication modules. The [1JAC] algorithm can then be used to obtain the singular value decomposition of the upper-triangular matrix $R$.

Hence, the singular value decomposition of an $m \times n$-matrix ($m \gg n$) $A$ (having rank $r$) defined by

$$A = U \Sigma V^T,$$

where $U^T U = V^T V = I_r$, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $\sigma_i > 0$ for $1 \leq i \leq r$, can be efficiently determined as follows:

## Block Householder-Jacobi [QJAC]

*STEP* 1: Apply block Householder reduction via (3.6) to the matrix $A$ to obtain the factorization

$$A = QR, \tag{23}$$

where $Q$ is $m \times n$ with orthonormal columns, and $R$ is upper triangular of order $n$.

*STEP* 2: Determine the SVD of the upper-triangular matrix via [1JAC],

$$R = \tilde{U} \begin{bmatrix} \tilde{\Sigma} \\ 0 \end{bmatrix} V^T, \tag{24}$$

where $\tilde{U}$ and $V$ are $n \times r$-matrices having orthogonal columns ($r \equiv \text{rank } A$) and $\tilde{\Sigma} = \text{diag } \sigma_i$ contains the $r$ nonzero singular values of $A$.

*STEP* 3: Recover the left singular vectors $u_i$ of $A$ by back-transforming the columns of $\tilde{U}$:

$$U = Q\tilde{U}, \tag{25}$$

where $Q$ is the product of the Householder transformations applied in Step 1 and $u_i$ is the $i$th column of $U$.

Note that in using [1JAC] for computing the SVD of $R$, we must iterate on a full $n \times n$-matrix which is initially upper-triangular. This sacrifice in storage must be made in order to capitalize upon the potential vectorization and parallelism inherent in [1JAC] on parallel machines with vector processors.

Charlier et al. [18] demonstrate that an implementation of Kogbetliantz's algorithm for computing the SVD of upper-triangular matrices is quite effective on a systolic array of processors. We recall that Kogbetliantz's method for computing the SVD of a real square matrix $A$ mirrors the [2JAC] method for symmetric matrices, in that the matrix $A$ is reduced to the diagonal form by an infinite sequence of plane rotations

$$A_{k+1} = U_k A_k V_k^T, \quad k = 1, 2, \ldots, \tag{26}$$

where $A_1 \equiv A$, and $V_k = V_k(i, j, \phi_{ij}^k)$, $U_k = U_k(i, j, \theta_{ij}^k)$ are orthogonal plane rotation matrices which deviate from $I_n$ and $I_m$, respectively, in the $(i,i)$-, $(j,j)$-, $(i,j)$- and $(j,i)$-entries. It follows that $A_k$ approaches the diagonal matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, where $\sigma_i$ is the $i$th singular value of $A$, and the products $(U_k \cdots U_2 U_1)$, $(V_k \cdots V_2 V_1)$ approach matrices whose $i$th column is the respective left and right singular vector corresponding to $\sigma_i$. For the case when the $\sigma_i$'s are not pathologically close, Paige and van Dooren [65] have shown that the row (or column) cyclic Kogbetliantz's method ultimately converges quadratically. For triangular matrices, Charlier and van Dooren [17] have demonstrated that Kogbetliantz's algorithm converges quadratically for those matrices having multiple or clustered singular values provided that singular values of the same cluster occupy adjacent diagonal position of $A_\nu$, where $\nu$ is the number of sweeps required for convergence. Even if we were to assume that $R$ in (22) satisfies this condition for quadratic convergence of the parallel Kogbetliantz's method in [65], the ordering of the rotations and subsequent row (or column) permutations needed to maintain the upper-triangular form is more efficient for systolic architectures than

for shared-memory parallel machines. One clear advantage of using [1JAC] to determine the SVD of $R$ lies in that the rotations defined by (15) or (16), as applied via the parallel ordering illustrated in Table 1, require no processor synchronization among any set of the $\lfloor \frac{1}{2}n \rfloor$ or $\lfloor \frac{1}{2}(n-1) \rfloor$ simultaneous plane rotations. The convergence rate of [1JAC], however, does not necessarily match that of Kogbetliantz's algorithm.

Let

$$R_k = D_k + E_k + E_k^T,$$

and

$$S_k = R_k^T R_k = \tilde{D}_k + \tilde{E}_k + \tilde{E}_k^T, \tag{27}$$

where $D_k$, $\tilde{D}_k$ are diagonal matrices and $E_k$, $\tilde{E}_k$ are strictly upper-triangular.

Although we cannot guarantee quadratic convergence for [1JAC], we can always produce clustered singular values on adjacent positions of $\tilde{D}_k$ for any matrix $A$. If we monitor the magnitudes of the elements of $\tilde{D}_k$ and $\tilde{E}_k$ in (28) for successive values of $k$ in [1JAC] (for clustered singular values), $S_k$ will converge to a diagonal form through an intermediate *block* diagonal form, where each of the principal submatrices (positioned along $\tilde{D}_k$) has diagonal elements which comprise one cluster of singular values of $A$ (see [18] and [11]). Thus, after a particular number of critical sweeps $k_{cr}$, we obtain

$$S_{k_{cr}} = \begin{bmatrix} \boxed{T_1} & & & & \\ & \boxed{T_2} & & & \\ & & \boxed{T_3} & & \\ & & & \ddots & \\ & & & & \boxed{T_{n_c}} \end{bmatrix} \tag{28}$$

so that the SVD of each $T_i$, $i = 1, 2, \ldots, n_c$, can be computed in parallel by either a Jacobi or Kogbetliantz's method. Each symmetric matrix $T_i$ will, in general, be dense of order $q_i$, representing the number of singular values of $A$ contained in the $i$th cluster. Since the quadratic convergence of Kogbetliantz's method for upper-triangular matrices [17] mirrors the quadratic convergence of the two-sided Jacobi method, [2JAC], for symmetric matrices having clustered spectra [97], we would obtain a faster global convergence for $k > k_{cr}$ if [2JAC], rather than [1JAC], were applied to each of the $T_i$. Thus, a *hybrid* method consisting of an initial phase of several [1JAC] iterations followed by [2JAC] on the resulting subproblems would combine the optimal parallelism of [1JAC] and the fast convergence of the [2JAC] method. Of course, the difficulty in implementing such a method lies in the determination of the critical number of sweeps $k_{cr}$. We note that such a hybrid SVD method would be quite suitable for implementation on multiprocessors with hierarchical memory structure and/or vector processors.

# 3 Methods for large and sparse matrices

## 3.1 Sparse storages and linear systems

When the matrix is large and sparse, a compact storage scheme must be considered. The principle behind such storage schemes is to store only the nonzero entries and sometimes more data as in band or profile storage schemes. For a more detailed presentation of various compact storage schemes, we refer for instance to [29, 76]. Here, we consider only the Compressed-Sparse-Row format (CSR) storage scheme.

Let us consider a sparse matrix $A$ of order $n$ with $n_z$ (denoted `nz` in algorithms) non-zero entries. The CSR format is organized into three one-dimensional arrays :

`array a(1:nz) :` contains all the nonzero entries of the matrix sorted by rows ; within a row no special ordering is assumed although it is often preferable to sort the entries by increasing column indices.

`array ja(1:nz) :` contains all the column indices of the nonzero entries in the same order as the order of the entries in array `a`.

`array ia(1:n+1) :` , `ia(i)` ($i = 1, \cdots, n$) is the index of the first nonzero entry of the i-th row which is stored in array `a` and `ia(n+1)` is set to $n_z + 1$.

The main procedures which use a matrix stored in that way are the multiplications of a matrix, $A$ or $A^T$, by a vector $x \in \mathbb{R}^n$. The corresponding algorithms are :

```
ALGORITHM : y := y + A*x
    for i = 1:n,
        for l = ia(i) : ia(i+1)-1,
            y(i) = y(i) + a(l)*x(ja(l)) ;
        end ;
    end ;
```

```
ALGORITHM : y := y + A^T*x
    for i = 1:n,
        for l = ia(i) : ia(i+1)-1,
            y(ja(l)) = y(ja(l)) + a(l)*x(i) ;
        end ;
    end
```

Solving linear systems which are defined by sparse matrices is not an easy task. One may consider direct methods, invariably consisting of matrix factorization, or consider iterative schemes. In direct solvers, care is needed to minimize the fill-in in the triangular factors, while in iterative methods, adopting effective preconditioning techniques is vital for fast convergence.

It is usually admitted that direct methods are more robust but are economical only when the triangular factors are not too dense, and when the size of the linear system is not too large. Reordering schemes are almost necessary to keep the level of fill-in as low as possible. Also, while pivoting strategies for dense linear systems are relaxed to minimize fill-in, the most effective sparse factorization schemes forbid the use of pivots below a given threshold. It is well known that the QR factorization schemes, one of the most robust, is not used often in direct solvers as it suffers from a high level of fill-in. Such a high level of fill-in occurs because the upper triangular factor, $R$, is the transpose of the Cholesky factor of matrix $A^T A$ which is much more dense than the original matrix $A$. Nevertheless, we shall see that this orthogonal factorization is a viable tool for computing the smallest singular value of sparse matrices. A survey of the state-of-the-art of sparse

matrix factorization may be found in the following references [2, 3, 12, 23, 28, 25, 29, 30, 35]. Current efficient packages for LU factorization include UMFPACK [22], SuperLU [52] and MUMPS [4].

When the order of the matrix is so large as to make the use of direct methods prohibitively expensive in storage and time, one resorts to iterative solvers. Classic iterative solvers such as the relaxation schemes methods of Jacobi, Gauss Seidel, SOR or SSOR, are easy to use but not as effective as Krylov subspace methods. The latter class uses the matrix only through the procedures of matrix-vector multiplications as defined above. Moreover, under certain conditions, Krylov subspace schemes exhibit superlinear convergence. Often, however, Krylov subspace schemes are successful only in conjunction with a preconditioning strategy. This is especially true for nonsymmetric ill-conditioned linear systems. Such preconditioners may be one of the above relaxation methods, approximate factorizations or approximate inverses of the matrix of coefficients. A state-of-the-art survey of iterative solvers may be found in the following references [5, 28, 6, 44, 60, 76] or other references there. When the matrix is symmetric positive definite, a preconditioned Conjugate Gradient scheme (PCG) may be an optimal choice as an iterative solver, [42]. For symmetric indefinite systems, methods like SYMMLQ and MINRES [66] are adequate, but surprisingly PCG is often used with great success even though it may fail in theory. For nonsymmetric systems, the situation is less clear since available iterative schemes cannot combine minimization of the residual, at any given step, for a given norm within the Krylov subspace, and orthogonalizing it w.r.t. the same subspace for some scalar product. Therefore, two classes of methods arise. The most popular methods include GMRES [77], Bi-CGSTAB [94], QMR [33] and TFQMR [32].

Before presenting methods for computing the sparse singular value decomposition, we note that classical methods for determining the SVD of dense matrices: the Golub-Kahan-Reinsch method [37], [40] and Jacobi-like SVD methods [11], [46] are not viable for large sparse matrices. Since these methods apply orthogonal transformations (Householder or Givens) directly to the sparse matrix $A$, they incur excessive fill-ins and thereby require tremendous amounts of storage. Another drawback to these methods is that they will compute all the singular triplets of $A$, and hence may be computationally wasteful when only a few of the smallest, or largest, singular triplets are desired. We demonstrate how canonical sparse symmetric eigenvalue problems can be used to (indirectly) compute the sparse singular value decomposition.

Since the computation of the smallest singular value is equivalent to computing an eigenvalue of a symmetric matrix, which is the augmented matrix or the matrix of the normal equations, in what follows we present methods that are specifically designed for such problems.

## 3.2 Subspace iteration [SISVD]

Subspace iteration is perhaps one of the simplest algorithms used to solve large sparse eigenvalueproblems. As discussed in [69], subspace iteration may be viewed as a block generalization of the classical power method. The basic version of subspace iteration was first introduced by Bauer [7] and if adapted to the matrix

$$\hat{B} = \gamma^2 I_n - A^T A, \tag{29}$$

would involve forming the sequence

$$Z_k = \hat{B}^k Z_0,$$

where $\gamma^2$ is chosen so that $\hat{B}$ is (symmetric) positive definite and $Z_0 = [z_1, z_2, \ldots, z_s]$ is an $n \times s$ matrix. If the column vectors, $z_i$, are normalized separately (as done in the Power method), then these vectors will converge to the dominant eigenvectors of $\hat{B}$, which are also the right singular vectors corresponding to the smallest singular values of $A$. Thus, the columns of the matrix $Z_k$ will progressively lose linear independence. In order to approximate the $p$-largest eigenpairs of $\hat{B}$, Bauer demonstrated that linear independence among the $z_i$'s could be maintained via reorthogonalization at each step, by a modified Gram-Schmidt procedure, for example. However, the convergence rate of the $z_i$'s to eigenvectors of $\hat{B}$ will only be linear.

The most sophisticated implementation of subspace iteration is that of Rutishauser's `RITZIT` (see [75]). This particular algorithm incorporates both a Rayleigh-Ritz procedure and acceleration via Chebyshev polynomials. The iteration which embodies the `RITZIT` program is given in Table 2. The Rayleigh Quotient matrix, $H_k$, in step (3) is essentially the projection of $\hat{B}^2$ onto the span $Z_{k-1}$. The three-term recurrence in step (6) follows from the adaptation of the Chebyshev polynomial of degree $q$, say $T_q(x)$, to the interval $[-e, e]$, where $e$ is chosen to be the smallest eigenvalue of $H_k$. This use of Chebyshev polynomials has the desired effects of damping unwanted eigenvalues of $\hat{B}$ and producing an improved rate of convergence which is considerably higher than the original rate of convergence, governed by $\theta_s/\theta_1$ ($\theta_1 \geq \theta_2 \geq \cdots \geq \theta_s$), where $\theta_i's$ are the eigenvalues of $H_k$, given by the square roots of the diagonal matrix $\Delta_k^2$ in step (4) of Table 2.

| | |
|---|---|
| (1) | Compute $C_k = \hat{B} Z_{k-1}$ |
| (2) | Factor $C_k = Q_k R_k$ |
| (3) | Form $H_k = R_k R_k^T$ |
| (4) | Factor $H_k = P_k \Delta_k^2 P_k^T$ |
| (5) | Form $Z_k = Q_k P_k$ |
| (6) | Iterate $Z_{k+j} = \frac{2}{e} \hat{B} Z_{k+j-1} - Z_{k+j-2}$ $(j = 2, \ldots, q)$ |

Table 2: Subspace iteration as implemented in Rutishauser's ritzit [`SISVD`].

We note that one could alternatively compute the eigenpairs of the positive definite 2-cyclic matrix

$$\tilde{B} = \begin{pmatrix} \gamma I & A \\ A^T & \gamma I \end{pmatrix}, \tag{30}$$

where $\gamma$ is an estimate of the largest singular value of $A$. The smallest singular values of $A$, in this case, will lie in the center of the spectrum of $\tilde{B}$ (see Section 1.1), and thus prohibit suppression of the unwanted (largest) singular values by the use of Chebyshev polynomials defined on the symmetric interval $[-e, e]$. Thus, it is preferable to approximate eigenpairs of $\hat{B} = \gamma^2 I_n - A^T A$ instead. In practice, $\gamma$ can be chosen as either $\| A \|_1$ or $\| A \|_\infty$, depending upon the sparse data structure used to store the nonzero elements (row or column-wise). Once, the eigenvectors of $\hat{B}$ (right singular vectors of $A$) have been determined, one can recover the left singular vectors $(u_i)$ of $A$ via $u_i = \frac{1}{\sigma_i} A v_i$ (see Section 1.1).

The orthogonal factorization in step (2) of Table 2 may be computed by a modified Gram-Schmidt procedure or by Householder transformations provided that the orthogonal matrix $Q_k$ is explicitly available for the computation of $Z_k$ in step (5). On multiprocessor architectures, especially those having hierarchical memories, one may achieve high performance by using either a block Gram-Schmidt [34] or the block Householder

orthogonalization method in step (2). To improve upon the two-sided Jacobi algorithm, originally suggested by Rutishauser [74] for the spectral decomposition in step (4) of `ritzit`, one may employ a parallel two- or one-sided Jacobi method on a multiprocessor. In fact, the one-sided Jacobi scheme, when appropriately adapted for symmetric positive definite matrices (see [11]), is quite efficient for step (4) provided the dimension of the current subspace, $s$, is not too large. For larger subspaces, an optimized implementation of the classical EISPACK [87] pair, TRED2 and TQL2, or Cuppen's algorithm as parallelized by Dongarra and Sorensen [27] may be used in step (4).

The success of Rutishauser's subspace iteration method using Chebyshev acceleration relies upon the following strategy for delimiting the degree of the Chebyshev polynomial, $T_q\left(\frac{x}{e}\right)$, on the interval $[-e, e]$, where $e = \theta_s$ (assuming $s$ vectors carried and $k = 1$ initially), $\xi_1 = 0.04$ and $\xi_2 = 4$:

$$q_{new} = \min\{2q_{old}, \hat{q}\}, \quad \text{where}$$

$$\hat{q} = \begin{cases} 1, & \text{if } \theta_1 < \xi_1\theta_s \\ 2 \times \max\left[\dfrac{\xi_2}{\operatorname{arccosh}\left(\frac{\theta_s}{\theta_1}\right)}, 1\right] & \text{otherwise.} \end{cases} \tag{31}$$

The polynomial degree of the current iteration is then taken to be $q = q_{new}$. It can easily be shown that the strategy in (31) insures that

$$\left\| T_q\left[\frac{\theta_1}{\theta_s}\right] \right\|_2 = \cosh\left[q \operatorname{arccosh}\left(\frac{\theta_1}{\theta_s}\right)\right] \leq \cosh(8) < 1500.$$

Although this bound has been quite successful for `ritzit`, we can easily generate several variations of polynomial-accelerated subspace iteration schemes (`SISVD`) using a more flexible bound. Specifically, we consider an *adaptive* strategy for selecting the degree $q$ in which $\xi_1$ and $\xi_2$ are treated as control parameters for determining the *frequency* and the *degree* of polynomial acceleration, respectively. In other words, large (small) values of $\xi_1$, inhibit (invoke) polynomial acceleration, and large (small) values of $\xi_2$ yield larger (smaller) polynomial degrees when acceleration is selected. Correspondingly, the number of matrix-vector multiplications will increase with $\xi_2$ and the total number of iterations may well increase with $\xi_1$. Controlling the parameters, $\xi_1$ and $\xi_2$, allows us to monitor the method's complexity so as to maintain an optimal balance between dominating kernels (e.g., sparse matrix multiplication, orthogonalization, and spectral decomposition). We will demonstrate these controls in the polynomial acceleration-based trace minimization SVD method discussed in Section 3.4.

## 3.3 Lanczos methods

### 3.3.1 The Single-vector Lanczos method [`LASVD`]

Other popular methods for solving large, sparse, symmetric eigenproblems originated from a method attributed to Lanczos (1950). This method generates a sequence of tridiagonal matrices $T_j$ with the property

that the extremal eigenvalues of the $j \times j$ matrix $T_j$ are progressively better estimates of the extremal eigenvalues of the original matrix $H$. Let us consider the $(m+n) \times (m+n)$ 2-cyclic matrix $B$ given in (3), where $A$ is the $m \times n$ matrix whose singular triplets are sought. Also, let $w_1$ be a randomly generated starting vector such that $\|w_1\|_2 = 1$. For $j = 1, 2, \ldots l$ define the corresponding Lanczos matrices $T_j$ using the following recursion [67]. Define $\beta_1 \equiv 0$ and $v_0 \equiv 0$, then for $i = 1, 2, \ldots, l$ define Lanczos vectors $w_i$ and scalars $\alpha_i$ and $\beta_{i+1}$ where

$$\begin{gathered} \beta_{i+1} w_{i+1} = B w_i - \alpha_i w_i - \beta_i w_{i-1}, \text{ and } \alpha_i = w_i^T (B w_i - \beta_i w_{i-1}), \\ |\beta_{i+1}| = \|B w_i - \alpha_i w_i - \beta_i w_{i-1}\|_2. \end{gathered} \tag{32}$$

For each $j$, the corresponding Lanczos matrix $T_j$ is defined as a real symmetric, tridiagonal matrix having diagonal entries $\alpha_i (1 \leq i \leq j)$, and subdiagonal (superdiagonal) entries $\beta_{i+1}$ $(1 \leq i \leq (j-1))$, i.e.,

$$T_j \equiv \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \beta_j \\ & & & & \beta_j & \alpha_j \end{pmatrix}. \tag{33}$$

By definition, the vectors $\alpha_i w_i$ and $\beta_i w_{i-1}$ in (32) are respectively, the orthogonal projections of $B w_i$ onto the most recent $w_i$ and $w_{i-1}$. Hence for each $i$, the next Lanczos vector $w_{i+1}$ is obtained by orthogonalizing $B w_i$ with respect to $w_i$ and $w_{i-1}$. The resulting $\alpha_i$, $\beta_{i+1}$ obtained in these orthogonalizations, define the corresponding Lanczos matrices. If we rewrite (32) in matrix form, then for each $j$ we have

$$B W_j = W_j T_j + \beta_{j+1} w_{j+1} e_j, \tag{34}$$

where $W_j \equiv [w_1, w_2, \ldots, w_j]$ is an $n \times j$ matrix whose $k$th column is the $k$th Lanczos vector, and $e_j$ is the $j$th column of the $(m+n) \times (m+n)$ identity matrix. Thus, the Lanczos recursion (34) generates a family of real symmetric tridiagonal matrices related to both $B$ and $w_1$. Table 3 outlines the basic Lanczos procedure for computing the eigenvalues and eigenvectors of the symmetric 2-cyclic matrix $B$.

As in subspace iteration, the matrix $B$ is only referenced through matrix-vector multiplication. At each iteration, the basic Lanczos recursion requires only the two most recently-generated vectors, although for finite-precision arithmetic modifications suggested by Grcar [43], Partlett and Scott [71], and Simon [84] require additional Lanczos vectors to be readily accessible via secondary storage. We note that on a multiprocessor architecture, step (2) in Table 3 may benefit from any available optimized library routine that solves the symmetric tridiagonal eigenvalue problem (e.g., multisectioning in [53] or divide and conquer in [27]).

In using finite-precision arithmetic, any practical Lanczos procedure must address problems created by losses in the orthogonality of the Lanczos vectors, $w_i$. Such problems include the occurrence of numerically-multiple eigenvalues of $T_j$ (for large $j$) for simple eigenvalues of $B$, and the appearance of spurious eigenvalues among the computed eigenvalues for some $T_j$. Approaches to deal with these problems range between two different extremes. The first involves total reorthogonalization of every Lanczos vector with respect to every

| | |
|---|---|
| (1) | Use any variant of the Lanczos recursion (32) to generate a family of real symmetric tridiagonal matrices, $T_j$ $(j = 1, 2, \ldots, q)$. |
| (2) | For some $k \leq q$, compute relevant eigenvalues of $T_k$. |
| (3) | Select some or all of these eigenvalues as approximations to the eigenvalues of the matrix $B$, and hence singular values of $A$. |
| (4) | For each eigenvalue $\lambda$ compute a corresponding unit eigenvector $z$ such that $T_k z = \lambda z$. Map such vectors onto the corresponding Ritz vectors $y \equiv W_q z$, which are then used as approximations to the desired eigenvectors of the matrix $B$ or the singular vectors of $A$. |

Table 3: Single-vector Lanczos recursion [LASVD].

previously-generated vector [38]. The other approach accepts the loss of orthogonality and deals with these problems directly. Total reorthogonalization is certainly one way of maintaining orthogonality, however, it will require additional storage and additional arithmetic operations. As a result, the number of eigenvalues which can be computed is limited by the amount of available secondary storage. On the other hand, a Lanczos procedure with no reorthogonalization needs only the two most recently-generated Lanczos vectors at each stage, and hence has minimal storage requirements. Such a procedure requires, however, the tracking [20] of the resulting spurious eigenvalues of $B$ (singular values of $A$) associated with the loss of orthogonality in the Lanczos vectors, $w_i$.

We employ a version of a single-vector Lanczos algorithm (32) equipped with a selective reorthogonalization strategy, LANSO, designed by Parlett and Simon, [71], [84]. This particular method (LASVD) is primarily designed for the standard and generalized symmetric eigenvalue problem. We simply apply it to either $H = A^T A$ or the 2-cyclic matrix $B$ defined in (3).

### 3.3.2 The Block Lanczos method [BLSVD]

Here, we consider a block analogue of the single vector Lanczos method. Exploiting the structure of the matrix $B$ in (3), we can obtain an alternative form for the Lanczos recursion (32). If we apply the Lanczos recursion specified by (32) to $B$ with a starting vector $\tilde{u} = (u, 0)^T$ such that $\|\tilde{u}\|_2 = 1$, then the diagonal entries of the real symmetric tridiagonal Lanzcos matrices generated are all identically zero. The Lanzcos recursion in (32) reduces to the following: define $u_1 \equiv u$, $v_0 \equiv 0$, and $\beta_1 \equiv 0$, then for $i = 1, 2, \ldots, k$

$$\begin{aligned} \beta_{2i} v_i &= A^T u_i - \beta_{2i-1} v_{i-1}, \\ \beta_{2i+1} u_{i+1} &= A v_i - \beta_{2i} u_i. \end{aligned} \tag{35}$$

The Lanczos recursion (35), however, can only compute the distinct singular values of an $m \times n$ matrix $A$ and not their multiplicities. Following the block Lanczos recursion for the sparse symmetric eigenvalue problem [93], [41], (35) can be represented in matrix form as

$$\begin{aligned} A^T \hat{U}_k &= \hat{V}_k J_k^T + Z_k, \\ A \hat{V}_k &= \hat{U}_k J_k + \hat{Z}_k, \end{aligned} \tag{36}$$

where $\hat{U}_k = [u_1, \ldots, u_k]$, $\hat{V}_k = [v_1, \ldots, v_k]$, $J_k$ is a $k \times k$ bidiagonal matrix with $J_k[j,j] = \beta_{2j}$ and $J_k[j,j+1] = \beta_{2j+1}$, and $Z_k$, $\tilde{Z}_k$ contain remainder terms. It is easy to show that the nonzero singular values of $J_k$ are the same as the positive eigenvalues of

$$K_k \equiv \begin{pmatrix} O & J_k \\ J_k^T & O \end{pmatrix}. \tag{37}$$

For the block analogue of (36), we make the simple substitutions

$$u_i \leftrightarrow U_i, \quad v_i \leftrightarrow V_i,$$

where $U_i$ is $m \times b$, $V_i$ is $n \times b$, and $b$ is the current block size. The matrix $J_k$ is now a block upper bidiagonal matrix of order $bk$

$$J_k \equiv \begin{pmatrix} S_1 & R_1^T & & & \\ & S_2 & R_2^T & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ & & & \cdot & R_{k-1}^T \\ & & & & S_k \end{pmatrix}, \tag{38}$$

where the $S_i$'s and $R_i$'s are $b \times b$ upper-triangular matrices. If $U_i$'s and $V_i$'s form mutually orthogonal sets of $bk$ vectors so that $\hat{U}_k$ and $\hat{V}_k$ are orthonormal matrices, then the singular values of the matrix $J_k$ will be identical to those of the original $m \times n$ matrix $A$. Given the upper block bidiagonal matrix $J_k$, we approximate the singular triplets of $A$ by first computing the singular triplets of $J_k$. To determine the left and right singular vectors of $A$ from those of $J_k$, we must retain the Lanczos vectors of $\hat{U}_k$ and $\hat{V}_k$. Specifically, if $\{\sigma_i^{(k)}, y_i^{(k)}, z_i^{(k)}\}$ is the $i$th singular triplet of $J_k$, then the approximation to the $i$th singular triplet of $A$ is given by $\{\sigma_i^{(k)}, \hat{U}_k y_i^{(k)}, \hat{V}_k z_i^{(k)}\}$, where $\hat{U}_k y_i^{(k)}$, $\hat{V}_k z_i^{(k)}$ are the left and right approximate singular vectors, respectively. The computation of singular triplets for $J_k$ requires two phases. The first phase reduces $J_k$ to a bidiagonal matrix $B_k$ having diagonal elements $\{\alpha_1, \alpha_2, \ldots, \alpha_{bk}\}$ and superdiagonal elements $\{\beta_1, \beta_2, \ldots, \beta_{bk-1}\}$ via a finite sequence of orthogonal transformations (thus preserving the singular values of $J_k$). The second phase reduces $B_k$ to diagonal form by a modified $QR$ algorithm. This diagonalization procedure is discussed in detail in [40]. The resulting diagonalized $B_k$ will yield the aproximate singular values of $A$, while the corresponding left and right singular vectors are determined through multiplications by all the left and right transformations used in both phases of the SVD of $J_k$.

There are a few options for the reduction of $J_k$ to the bidiagonal matrix, $B_k$. Golub, Luk, and Overton in [39] advocated the use of either band Householder or band Givens methods which in effect *chase off* (or zero) elements on the diagonals above the first super-diagonal of $J_k$. In either reduction (bi-diagonalization or diagonalization), the computations are primarily sequential and offer limited data locality or parallelism for possible exploitation on a multiprocessor. For this reason, we adopt the single vector Lanczos bidiagonalization recursion defined by (35) and (36) as our strategy for reducing the upper block bidiagonal matrix $J_k$ to the bidiagonal form $(B_k)$, i.e.,

$$\begin{aligned} J_k^T \hat{Q} &= \hat{P} B_k^T, \\ J_k \hat{P} &= \hat{Q} B_k, \end{aligned} \tag{39}$$

or

$$\begin{array}{rcl} J_k p_j & = & \alpha_j q_j + \beta_{j-1} q_{j-1}, \\ J_k^T q_j & = & \alpha_j p_j + \beta_j p_{j+1}, \end{array} \tag{40}$$

where $\hat{P} \equiv \{p_1, p_2, \ldots, p_{bk}\}$ and $\hat{Q} \equiv \{q_1, q_2, \ldots, q_{bk}\}$ are orthonormal matrices of order $bk \times bk$. The recursions in (40) require band matrix-vector multiplications which can be easily exploited by optimized level-2 BLAS routines [26] now resident in optimized mathematical libraries on most high-performance computers. For orthogonalization of the outermost Lanczos vectors, $\{U_i\}$ and $\{V_i\}$, as well as the innermost Lanczos vectors, $\{p_i\}$ and $\{q_i\}$, we have chosen to apply a complete or total reorthogonalization [38] strategy to insure robustness in our triplet approximations for the matrix $A$. This hybrid Lanczos approach which incorporates inner iterations of single-vector Lanczos bidiagonalization within the outer iterations of a block Lanczos SVD recursion is also discussed in [10].

As an alternative to the outer recursion defined by (36), which is derived from the equivalent 2-cyclic matrix $B$, Table 4 depicts the simplified outer block Lanczos recursion for approximating the eigensystem of $A^T A$. Combining the equations in (36), we obtain

$$A^T A \hat{V}_k = \hat{V}_k H_k,$$

where $H_k = J_k^T J_k$ is the $k \times k$ symmetric block tridiagonal matrix

$$H_k \equiv \begin{pmatrix} S_1 & R_1^T & & & & \\ R_1 & S_2 & R_2^T & & & \\ & R_2 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & R_{k-1}^T \\ & & & & R_{k-1} & S_k \end{pmatrix}, \tag{41}$$

having block size $b$. We then apply the block Lanczos recursion [38] in Table 4 for computing the eigenpairs of the $n \times n$ symmetric positive definite matrix $A^T A$. The tridiagonalization of $H_k$ via an inner Lanczos recursion follows from simple modifications of (34). Analogous to the reduction of $J_k$ in (38), the computation of eigenpairs of the resulting tridiagonal matrix can be performed via a Jacobi or QR-based symmetric eigensolver.

As with the previous iterative SVD methods, we access the sparse matrices $A$ and $A^T$ for this hybrid Lanczos method only through sparse matrix-vector multiplications. Some efficiency, however, is gained in the outer (block) Lanczos iterations by the multiplication of $b$ vectors rather than by a single vector. These dense vectors may be stored in a fast local memory (cache) of a hierarchical memory-based architecture, and thus yield more effective data reuse. A stable variant of Gram-Schmidt orthogonalization [75], which requires efficient dense matrix-vector multiplication (level-2 BLAS) routines [26] or efficient dense matrix-matrix multiplication (level-3 BLAS) routines [50], is used to produce the orthogonal projections of $Y_i$ (i.e., $R_{i-1}$) and $W_i$ (i.e., $S_i$) onto $\tilde{V}^\perp$ and $\tilde{U}^\perp$, respectively, where $U_0$ and $V_0$ contain converged left and right singular vectors, respectively, and

$$\tilde{V} = (V_0, V_1, \ldots, V_{i-1}) \text{ and } \tilde{U} = (U_0, U_1, \ldots, U_{i-1}).$$

| | |
|---|---|
| (1) | [Formation of symmetric block tridiagonal matrix $H_k$.] |
| | Choose $V_1$ ($n \times b$ and orthonormal) and $c = max\{bk\}$. |
| | Compute $S_1 = V_1^T A^T A V_1$. ($V_0, R_0^T = 0$ initially). |
| | For $i = 2, 3, \ldots, k$ do: ($k = \lfloor c/b \rfloor$) |
| (2) | Compute $Y_{i-1} = A^T A V_{i-1} - V_{i-1} S_{i-1} - V_{i-1} R_{i-2}^T$. |
| (3) | Orthogonalize $Y_{i-1}$ against $\{V_\ell\}_{\ell=0}^{i-1}$ |
| (4) | Factor $Y_{i-1} = V_i R_{i-1}$. |
| (5) | Compute $S_i = V_i^T A^T A V_i$. |

Table 4: Hybrid Lanczos outer iteration [BLSVD].

## 3.4 The Trace Minimization method [TRSVD]

Another candidate subspace method for the SVD of sparse matrices is based upon the trace minimization algorithm discussed in [81] for the generalized eigenvalue problem

$$Hx = \lambda Gx, \tag{42}$$

where $H$ and $G$ are symmetric and $G$ is also positive definite. In order to compute the SVD of an $m \times n$ matrix $A$, we initially replace $H$ with $\tilde{B}$ as defined in (30) or set $H = A^T A$. Since we need only consider equivalent standard symmetric eigenvalue problems, we simply define $G = I_{m+n}$ (or $I_n$ if $H = A^T A$). Without loss of generality, let us assume that $H = A^T A$, $G = I_n$ and consider the associated symmetric eigensystem of order $n$. If $\mathcal{Y}$ is defined as the set of all $n \times p$ matrices $Y$ for which $Y^T Y = I_p$, then using the Courant-Fischer theorem (see [96]) we obtain

$$\min_{Y \in \mathcal{Y}} \text{trace}(Y^T H Y) = \sum_{i=1}^{p} \tilde{\sigma}_{n-i+1}, \tag{43}$$

where $\sqrt{\tilde{\sigma}_i}$ is a singular value of $A$, $\lambda_i = \tilde{\sigma}_i$ is an eigenvalue of $H$, and $\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_n$. In other words, given an $n \times p$ matrix $Y$ which forms a *section* of the eigenvalue problem

$$Hz = \lambda z, \tag{44}$$

i.e.,

$$Y^T H Y = \tilde{\Sigma}, \quad Y^T Y = I_p, \tag{45}$$

$$\tilde{\Sigma} = diag(\tilde{\sigma}_n, \tilde{\sigma}_{n-1}, \ldots, \tilde{\sigma}_{n-p+1}),$$

our trace minimization scheme [TRSVD] [81], see also [9], finds a sequence of iterates $Y_{k+1} = F(Y_k)$, where both $Y_k$ and $Y_{k+1}$ form a section of (44), and have the property $\text{trace}(Y_{k+1}^T H Y_{k+1}) < \text{trace}(Y_k^T H Y_k)$. From (43), the matrix $Y$ in (45) which minimizes $\text{trace}(Y^T H Y)$ is the matrix of $H$-eigenvectors associated with

the $p$-smallest eigenvalues of the problem (44). As discussed in [81], $F(Y)$ can be chosen so that global convergence is assured. Moreover, (45) can be regarded as the quadratic minimization problem

$$\text{minimize} \quad \text{trace}(Y^T H Y) \tag{46}$$

subject to the constraints

$$Y^T Y = I_p. \tag{47}$$

Using Lagrange multipliers, this quadratic minimization problem leads to solving the $(n+p)\times(n+p)$ system of linear equations

$$\begin{pmatrix} H & Y_k \\ Y_k^T & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L \end{pmatrix} = \begin{pmatrix} HY_k \\ 0 \end{pmatrix}, \tag{48}$$

so that $Y_{k+1} \equiv Y_k - \Delta_k$ will be an optimal subspace iterate.

Since the matrix $H$ is positive definite, one can alternatively consider the p independent (parallel) subproblems

$$\text{minimize} \quad ((y_j^{(k)} - d_j^{(k)})^T H (y_j^{(k)} - d_j^{(k)})) \tag{49}$$

subject to the constraints

$$Y^T d_j^{(k)} = 0, \quad j = 1, 2, \ldots, p,$$

where $d_j^{(k)} = \Delta_k e_j$, $e_j$ being the $j$th column of the identity, and $Y_k = [y_1^{(k)}, y_2^{(k)}, \ldots, y_p^{(k)}]$. The corrections $\Delta_k$ in this case are selected to be orthogonal to the previous estimates $Y_k$, i.e., so that (see [54])

$$\Delta_k^T Y_k = 0.$$

We then recast (48) as

$$\begin{pmatrix} H & Y_k \\ Y_k^T & 0 \end{pmatrix} \begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix} = \begin{pmatrix} Hy_j^{(k)} \\ 0 \end{pmatrix}, \quad j = 1, 2, \ldots, p, \tag{50}$$

where $l$ is a vector of order $p$ reflecting the Lagrange multipliers.

The solution of the $p$ systems of linear equations in (50) can be done in parallel by either a direct or iterative solver. Since the original matrix $A$ is assumed to be large, sparse and without any particular sparsity structure (pattern of nonzeros) we have chosen an iterative method (conjugate gradient) for the systems in (50). As discussed in [81] and [10], a major reason for using the conjugate gradient (CG) method for the solution of (50) stems from the ability to terminate CG iterations early without obtaining fully-accurate corrections $d_j^{(k)}$ that are more accurate than warranted. In later stages, however, as $Y_k$ converges to the desired set of eigenvectors of $B$, one needs full accuracy in computing the correction matrix $\Delta_k$.

### 3.4.1 Polynomial acceleration techniques for [TRSVD]

The Chebyshev acceleration strategy used within subspace iteration (see Section 3.2), can also be applied to [TRSVD]. However, to dampen unwanted (largest) singular values of $A$ in this context, we must solve the generalized eigenvalue problem

$$x = \frac{1}{P_q(\lambda)} P_q(H)x, \tag{51}$$

where $P_q(x) = T_q(x) + \epsilon I_n$, $T_q(x)$ is the Chebyshev polynomial of degree $q$ and $\epsilon$ is chosen so that $P_q(H)$ is (symmetric) positive definite. The appropriate quadratic minimization problem similar to (49) here can be expressed as

$$\text{minimize} \quad ((y_j^{(k)} - d_j^{(k)})^T (y_j^{(k)} - d_j^{(k)})) \tag{52}$$

subject to the constraints

$$Y^T P_q(H) d_j^{(k)} = 0, \quad j = 1, 2, \ldots, p.$$

In effect, we approximate the smallest eigenvalues of $H$ as the *largest* eigenvalues of the matrix $P_q(H)$ whose gaps are considerably larger than those of the eigenvalues of $H$.

Although the additional number of sparse matrix-vector multiplications associated with the multiplication by $P_q(H)$ will be significant for high degrees $q$, the system of equations via Lagrange multipliers in (50) becomes much easier to solve, i.e.,

$$\begin{pmatrix} I & P_q(H)Y_k \\ Y_k^T P_q(H) & 0 \end{pmatrix} \begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix} = \begin{pmatrix} y_j^{(k)} \\ 0 \end{pmatrix}, \quad j = 1, 2, \ldots, p. \tag{53}$$

It is easy to show that the updated eigenvector approximation, $y_j^{(k+1)}$, is determined by

$$y_j^{(k+1)} = y_j^{(k)} - d_j^{(k)} = P_q(H)Y_k[Y_k^T P_q^2(H) Y_k]^{-1} Y_k^T P_q(H) y_j^{(k)}.$$

Thus, we may not need to use an iterative solver for determining $Y_{k+1}$ since the matrix $[Y_k^T P_q^2(H) Y_k]^{-1}$ is of relatively small order $p$. Using the orthogonal factorization

$$P_q(H)Y_k = \hat{Q}\hat{R},$$

we have

$$[Y_k^T P_q^2(H) Y_k]^{-1} = \hat{R}^{-T}\hat{R}^{-1},$$

where the polynomial degree, $q$, is determined by the strategy defined in Section 3.2.

### 3.4.2 Shifting strategy for [TRSVD]

As discussed in [81], we can also accelerate the convergence of the $Y_k$'s to eigenvectors of $H$ by incorporating Ritz shifts (see [69]) into TRSVD. Specifically, we modify the symmetric eigenvalue problem in (44) as follows,

$$(H - \nu_j^{(k)} I) z_j = (\lambda_j - \nu_j^{(k)}) z_j, \quad j = 1, 2, \ldots, s, \tag{54}$$

where $\nu_j^{(k)} = \tilde{\sigma}_{n-j+1}^{(k)}$ is the $j$th approximate eigenvalue at the $k$th iteration of [TRSVD], with $\lambda_j$, $z_j$ being an exact eigenpair of $H$. In other words, we simply use our most recent approximations to the eigenvalues of $H$ from our $k$th section within [TRSVD] as Ritz shifts. As was shown by Wilkinson in [98], the Rayleigh quotient iteration associated with (54) will ultimately achieve cubic convergence to the square of an exact singular value $A$, $\sigma_{n-j+1}^2$, provided $\nu_j^{(k)}$ is sufficiently close to $\sigma_{n-j+1}^2$. However, since we have $\nu_j^{(k+1)} < \nu_j^{(k)}$ for all $k$, i.e., we approximate eigenvalues of $H$ from above, $H - \nu_j^{(k)} I$ will not be positive definite and thus we cannot guarantee the convergence of this shifted [TRSVD] method for any particular singular triplet of $A$. However, the strategy outlined in [81] has been quite successful in maintaining global convergence with shifting.

Table 5 outlines the basic steps of [TRSVD]. The scheme appropriately utilizes polynomial (Chebyshev) acceleration prior to the Ritz shifts. It is important to note that once shifting has been invoked (Step (4)) we abandon the use of Chebyshev polynomials $P_q(H)$ and solve shifted systems ($H$ replaced by $H - \nu_j^{(k)} I$) of the form (48) via (50) and the CG algorithm. The *context switch* from either non-accelerated (or polynomial-accelerated trace minimization iterations) to trace minimization iterations with Ritz shifting, is accomplished by monitoring the reduction of the residuals in (4) for isolated eigenvalues ($r_j^{(k)}$) or clusters of eigenvalues ($R_j^{(k)}$).

| | |
|---|---|
| (Step 0) | Set $k = 0$, choose an initial $n \times s$ subspace iterate $Y_0 = [y_1^{(0)}, y_2^{(0)}, \ldots, y_s^{(0)}]$. |
| (Step 1) | Form a section as in (45), or determine $Y_k$ such that $Y_k^T P_q(H) Y_k = I_p$, $Y_k^T Y_k = \tilde{\Sigma}$. |
| (Step 2) | Compute residuals: $r_j^{(k)} = H y_j^{(k)} - \tilde{\sigma}_{n-j+1}^{(k)} y_j^{(k)}$, and access accuracy. |
| (Step 3) | Analyze the current approximate spectrum (Gershgorin disks). |
| (Step 4) | Invoke Ritz shifting strategy ([81]):<br>For isolated eigenvalues:<br>if $\|r_j^{(k)}\|_2 \leq \eta \|r_j^{(k_0)}\|_2$, where $\eta \in [10^{-3}, 10^0]$ and $k_0 < k$ for some $j$.<br>For a cluster of eigenvalues (size $c$):<br>if $\|R_j^{(k)}\|_F \leq \eta \|R_j^{(k_0)}\|_F$, where $R_j^{(k)} \equiv \{r_j^{(k)}, \ldots, r_{j+c}^{(k)}\}$ and $k_0 < k$ for some $j$.<br>(Disable polynomial acceleration if shifting is selected.) |
| (Step 5) | Deflation: reduce subspace dimension, $s$, by number of $H$-eigenpairs accepted. |
| (Step 6) | Adjust polynomial degree $q$ via (31) for $P_q(H)$ in iteration $k+1$ (if needed). |
| (Step 7) | Update subspace iterate $Y_{k+1} \equiv Y_k - \Delta_k$ via (48) or (50). |
| (Step 8) | Set $k = k + 1$ and go to (Step 1). |

Table 5: [TRSVD] algorithm with Chebyshev acceleration and Ritz shifts.

The Chebyshev acceleration and Ritz shifting within [TRSVD] for approximating the the smallest singular value of a $374 \times 82$ matrix with only 1343 nonzero elements has cut down the number of iterations needed by [TRSVD] with no acceleration by a factor of 3, using the same stopping criteria. The convergence rate of [TRSVD] with Chebyshev acceleration plus Ritz shifts ($CA + RS$) is 3 times higher than [TRSVD] with no acceleration.

## 3.5 Refinement of Left Singular Vectors

As discussed in Section 1.1, since the smallest singular values of the matrix $A$ lie in the interior of the spectrum of either the 2-cyclic matrix $B$ (3) or the shifted matrix $\tilde{B}$ (30), all four candidate methods will have difficulties approximating these singular values. The Lanczos-based methods, [LASVD] and [BLSVD] cannot be expected to effectively approximate the interior eigenvalues of either $B$ or $\tilde{B}$. Similarly, subspace iteration [SISVD] would not be able to suppress the unwanted (largest) singular values if Chebyshev polynomials are defined on symmetric intervals of the form $[-e, e]$. [TRSVD], however, via direct or iterative schemes for solving the systems in (50), could indeed obtain accurate approximation of the smallest singular triplets.

In order to target the $p$-smallest singular triplets of $A$, we compute the $p$-smallest eigenvalues and eigenvectors of the (operator) matrices listed in Table 6 for LASVD, BLSVD, and TRSVD. With SISVD, we determine the $p$-largest eigenpairs of $\gamma^2 I_n - A^T A$. As discussed in Section 1.1, we may determine the singular values, $\sigma_i$, of $A$ and their corresponding right singular vectors, $v_i$, as eigenpairs of either $A^T A$ or $\gamma^2 I_n - A^T A$, where $A$ is $m \times n$ and $\gamma = \|A\|_{1,\infty}$. The corresponding left singular vector, $u_i$, must then be determined by

$$u_i = \frac{1}{\sigma_i} A v_i, \tag{55}$$

which may not be of sufficient accuracy (see Section 1.1) for a given precision in the residual (4). We point out, however, that alternative operators of the form (*shift and invert*)

$$(A^T A - \tilde{\sigma}^2 I)^{-1},$$

where $\tilde{\sigma}$ is a good approximation to an exact singular value of $A$, can also be used to determine the $p$-smallest singular values of $A$ (see [69], [75]). The effectiveness of this approach depends on an *accurate* approximation of the Cholesky factorization of $A^T A$, or preferably of the QR factorization of $A$ when $A$ is large and sparse rectangular matrix.

| Method | Label | Operator |
|---|---|---|
| Single-Vector Lanczos | LASVD | $A^T A$ |
| Block Lanczos | BLSVD | $A^T A$ |
| Subspace Iteration | SISVD | $\gamma^2 I_n - A^T A$ |
| Trace Minimization | TRSVD | $A^T A$ |

Table 6: Operators of equivalent symmetric eigenvalue problems for computing the $p$-smallest singular triplets of the $m \times n$ matrix $A$.

For [LASVD], we simply define $\tilde{B} = A^T A$ in (32) and apply the strategy outlined in Table 3. For [SISVD] we make the substitution $\tilde{B} = \gamma^2 I_n - A^T A$ for (30) and apply the method in Table 2. Similarly, we solve the generalized eigenvalue problem in (44) with

$$H = A^T A, \quad G = I_n,$$

and apply the strategy in Table 5 for the appropriate [TRSVD] method. Here, we do not require the shift, $\gamma^2$, since the trace minimization will converge to the smallest eigenvalues of $A^TA$ (and hence squares of singular values of $A$) by default. For [BLSVD], we combine the equations in (36) and obtain

$$A^TA\hat{V}_k = \hat{V}_kH_k + \hat{Z}_k,$$

where $H_k = J_k^TJ_k$ is the $k \times k$ symmetric block tridiagonal matrix in (41), with the $n \times k$ matrix $\hat{Z}_k$ containing the remainder terms. The appropriate block Lanczos (outer) recursion is then given by Table 4.

Having determined approximate singular values, $\tilde{\sigma}_i$ and corresponding right singular vectors, $\tilde{v}_i$, to a user-specified tolerance for the residual

$$\hat{r}_i = A^TA\tilde{v}_i - \tilde{\sigma}_i^2\tilde{v}_i, \tag{56}$$

we must then obtain an approximation to the corresponding left singular vector, $u_i$, via (55). As mentioned in Section 1.1, it is quite possible that square roots of the approximate eigenvalues of either $A^TA$ or $\gamma^2I_n - A^TA$ will be poor approximations to exact singular values of $A$ which are extremely small. This phenomenon, of course, will lead to poor approximations to the left singular vectors. Even if $\tilde{\sigma}_i$ (computed by any of the four methods) is an acceptable singular value approximation, the residual corresponding to the singular triplet $\{\tilde{\sigma}_i, \tilde{u}_i, \tilde{v}_i\}$, defined by (4), will be bounded by

$$\|r_i\|_2 \leq \|\hat{r}_i\|_2/[\tilde{\sigma}_i([\|\tilde{u}_i\|_2^2 + \|\tilde{v}_i\|_2^2)^{\frac{1}{2}}], \tag{57}$$

where $\hat{r}_i$ is the residual given in (56) for the symmetric eigenvalue problem for $A^TA$ or $(\gamma^2I_n - A^TA)$. Scaling by $\tilde{sigma}_i$ can easily lead to significant loss of accuracy in estimating the triplet residual norm, $\|r_i\|_2$, especially when $\tilde{\sigma}_i$ approaches the machine unit roundoff error, $\mu$.

One remedy is to refine the initial approximation of the left singular vector via (inverse iteration)

$$AA^T\tilde{u}_{i+1}^{(k)} = \tilde{\sigma}_i^2\tilde{u}_i^{(k)}, \tag{58}$$

where $\tilde{u}_i^{(0)} \equiv u_i$ from (55). Since $AA^T$ is symmetric semi-definite, direct methods developed by Aasen [1], Bunch and Kaufmann [16], or Parlett and Reid [70] could be used in each step of (58) if $AA^T$ is explicitly formed. With regard to iterative methods, the SYMMLQ algorithm developed by Paige and Saunders [68] can be used to solve these symmetric indefinite systems of equations. As an alternative, we consider the following equivalent eigensystem for the SVD of an $m \times n$ matrix $A$.

$$\begin{pmatrix} \gamma I_n & A^T \\ A & \gamma I_m \end{pmatrix} \begin{pmatrix} v_i \\ u_i \end{pmatrix} = (\gamma + \sigma_i) \begin{pmatrix} v_i \\ u_i \end{pmatrix}, \tag{59}$$

where $\{\sigma_i, u_i, v_i\}$ is the $i$th singular triplet of $A$ and

$$\gamma = min[1, max\{\sigma_i\}]. \tag{60}$$

One possible refinement recursion (inverse iteration) is thus given by

$$\begin{pmatrix} \gamma I_n & A^T \\ A & \gamma I_m \end{pmatrix} \begin{pmatrix} \tilde{v}_i^{(k+1)} \\ \tilde{u}_i^{(k+1)} \end{pmatrix} = (\gamma + \tilde{\sigma}_i) \begin{pmatrix} \tilde{v}_i^{(k)} \\ \tilde{u}_i^{(k)} \end{pmatrix}, \tag{61}$$

$$(\alpha I_n + \beta A^T A)\tilde{u}_i = (\alpha + \beta\tilde{\sigma}_i^2), \tilde{v}_i$$

| | |
|---|---|
| (1) | Solve eigensystems of the form $i = 1, 2, \ldots, p$, where $\alpha = 0$ and $\beta = 1$ for `LASVD`, `BLSVD`, and `TRSVD`, or $\alpha = \gamma^2 > \sigma_{max}^2$ and $\beta = -1$ for `SISVD`, |
| (2) | Define $\tilde{u}_i^{(0)} \equiv \frac{1}{\tilde{\sigma}_i} A\tilde{v}_i$, $i = 1, 2, \ldots, p$ . |
| (3) | For $k = 0, 1, 2, \ldots$ (until $\|r_i\|_2 \leq$ tolerance or $k \geq k_{max}$). Solve $\left(\gamma I_m - \frac{1}{\gamma} AA^T\right) \tilde{u}_i^{(k+1)} = (\gamma - \tilde{\sigma}_i^2/\gamma)\tilde{u}_i^{(k)}$, $i = 1, 2, \ldots, p$. Set $\tilde{u}_i^{(k+1)} = \tilde{u}_i^{(k+1)}/\|\tilde{u}_i^{(k+1)}\|_2$. |

Table 7: Refinement procedure for the left singular vector approximations obtained via scaling.

By applying block Gaussian elimination to (61) we obtain a more optimal form (reduced system) of the recursion

$$\begin{pmatrix} \gamma I_n & A^T \\ 0 & \gamma I_m - \frac{1}{\gamma} AA^T \end{pmatrix} \begin{pmatrix} \tilde{v}_i^{(k+1)} \\ \tilde{u}_i^{(k+1)} \end{pmatrix} = (\gamma + \tilde{\sigma}_i) \begin{pmatrix} \tilde{v}_i^{(k)} \\ \tilde{u}_i^{(k)} - \frac{1}{\gamma} A\tilde{v}_i^{(k)} \end{pmatrix}. \tag{62}$$

Our iterative refinement strategy for an approximate singular triplet of $A$, $\{\tilde{\sigma}_i, \tilde{u}_i, \tilde{v}_i\}$, is then defined by the last $m$ equations of (62), i.e.,

$$\left(\gamma I_m - \frac{1}{\gamma} AA^T\right) \tilde{u}_i^{(k+1)} = (\gamma + \tilde{\sigma}_i) \left(\tilde{u}_i^{(k)} - \frac{1}{\gamma} A\tilde{v}_i\right), \tag{63}$$

where the superscript $k$ is dropped from $\tilde{v}_i$ since we refine only our left singular vector approximation, $\tilde{u}_i$. If $\tilde{u}_i^{(0)} \equiv u_i$ from (59), then (63) can be rewritten as

$$\left(\gamma I_m - \frac{1}{\gamma} AA^T\right) \tilde{u}_i^{(k+1)} = (\gamma - \tilde{\sigma}_i^2/\gamma)\tilde{u}_i^{(k)}, \tag{64}$$

with (normalization)

$$\tilde{u}_i^{(k+1)} = \tilde{u}_i^{(k+1)}/\|\tilde{u}_i^{(k+1)}\|_2.$$

It is easy to show that the left-hand-side matrix in (63) is symmetric positive definite provided (60) holds. Accordingly, we may use parallel conjugate gradient iterations to refine each singular triplet approximation. Hence, the refinement procedure outlined in Table 6 may be considered as a *black box* procedure to follow the eigensolution of $A^T A$ or $\gamma^2 I_n - A^T A$ by any one of our four candidate methods for the sparse SVD. The iterations in step (3) of the refinement scheme in Table 7 terminate once the norms of the residuals of all $p$ approximate singular triplets ($\|r_i\|_2$) fall below a user-specified tolerance or after $k_{max}$ iterations.

**Summary**

In this work, we demonstrated that a trace minimization strategy [TRSVD] using Chebyshev acceleration, Ritz shifting, and an iterative refinement method for improving left singular vector approximations, can be quite economical and robust when computing several of the smallest singular triplets of sparse matrices arising from a vareity of applications. A single-vector Lanczos method [LASVD] can be quite competitive in speed compared to [TRSVD] at the risk of missing some of the desired triplets due to parameter selection difficulties. Whereas [TRSVD] is effective in achieving high accuracy in approximating all the desired singular triplets. [LASVD] is acceptable only when moderate to low accurate triplets are needed. A subspace iteration-based method [SISVD] using Chebyshev acceleration performed poorly for clustered singular values when polynomials of relatively high degree are used. Typically, this scheme requires less memory than the other three methods. A block Lanczos SVD method [BLSVD], on the other hand, is quite robust for obtaining all the desired singular triplets at the expense of large memory requirements. The use of alternative re-orthogonalization strategies could be a remedy for this difficulty. Finally, we have proposed a hybrid [TRSVD] method which circumvents the potential loss of accuracy when solving eigensystems of $A^T A$.

## 3.6 The Davidson methods

### 3.6.1 General framework of the methods

In 1975, Davidson [21]introduced a method for computing the smallest eigenvalues of the Schrödinger operator. Later, the method was generalized in two papers [63] and [19] in which the convergence of the method is proved. More recently, another version of the algorithm, under the name Jacobi-Davidson, was introduced in [86]. We present here a general framework for the class of Davidson methods, and point out how the various versions differ from one another. We should point out that for symmetric eigenvalue problems, the Davidson method version in [86] is strongly related to the trace minimization scheme discussed in Section 3.4, see [81] and [80]. All versions of the Davidson method may be regarded as various forms of preconditioning the basic Lanczos method. To illustrate this point, let us consider a symmetric matrix $A \in \mathbb{R}^{n\times n}$. Both classes of algorithms generate, at some iteration $k$, an orthonormal basis $V_k = [v_1, \cdots, v_k]$ of a $k$-dimensional subspace $\mathcal{V}_k$ of $\mathbb{R}^{n\times n}$. In the Lanczos algorithm, $\mathcal{V}_k$ is a Krylov subspace, but for Davidson methods, this is not the case. In both classes, however, the interaction matrix is given by the symmetric matrix $H_k = {V_k}^T A V_k \in \mathbb{R}^{k\times k}$. Likely with the Lanczos method, the goal is to obtain $V_k$ such that some eigenvalues of $H_k$ are good approximations of some eigenvalues of $A$ : if $(\lambda, y)$ is an eigenpair of $H_k$, then it is expected that the Ritz pair $(\lambda, x)$, where $x = V_k y$, is a good approximation of an eigenpair of $A$. Note that this occurs only for some eigenpairs of $H_k$ and at convergence.

Davidson methods differ from Lanczos in the definition of the new direction $w$ which will be incorporated in the subspace $\mathcal{V}_k$ to obtain $\mathcal{V}_{k+1}$. For Lanczos schemes the vector $w$ is given by $w = Av_k$, whereas for Davidson methods, a local improvement of the direction of the Ritz vector towards the sought after eigenvector is obtained by a quasi-Newton step (similar to the trace minimization scheme in [81] and [80]). In Lanczos, the following vector $v_{k+1}$ is computed by the three term recursion, if reorthogonalization is not considered, whereas in Davidson methods, the next vector is obtained by reorthogonalizing $w$ with respect to $\mathcal{V}_k$. Moreover, in this case, the matrix $H_k$ is no longer tridiagonal. Therefore, one iteration of Davidson

methods involves more arithmetic operations than the basic Lanczos scheme; it is at least as expensive as a Lanczos scheme with full reorthogonalization. Moreover, the basis $V_k$ must be stored which implies the need for limiting the maximum value $k_{\max}$ in order to control storage requirements. Consequently an algorithm with periodic restarts must be implemented.

To compute the smallest eigenvalue of matrix $A$, the skeleton of the basic algorithm is :

```
ALGORITHM : Generic Davidson
Choose an initial normalized vector V_1 = [v_1]
Repeat
    for k = 1 : kmax,
        compute W_k = A V_k ;
        compute the interaction matrix H_k = V_k^T A V_k;
        compute the smallest eigenpair (λ_k, y_k) of H_k ;
        compute the Ritz vector x_k = V_k y_k ;
        compute the residual r_k = W_k y_k − λ_k x_k ;
        if convergence then exit ;
        compute the new direction t_k ; (Davidson correction)
        V_{k+1} = MGS(V_k, t_k) ;
    end ;
    until convergence ;
    V_1 = MGS(x_{k,1}, t_k) ;
end repeat ;
```

Algorithms of the Davidson family differ only in how the vector $t_k$ is determined. Note that the above generic algorithm is expressed in a compact form, the steps which compute $W_k$ or $H_k$, however, only determine updates to $W_{k-1}$ or $H_{k-1}$. In this algorithm MGS denotes the Modified Gram-Schmidt algorithm which is used to orthogonalize $t_k$ with respect to $V_k$ and then to normalize the resulting vector to obtain $v_{k+1}$.

Similar to the Lanczos algorithm, a block version of Davidson methods may be considered for approximating the $s$ smallest eigenvalues of $A$. These are even more closely related to the trace minimization scheme [81] and [80].

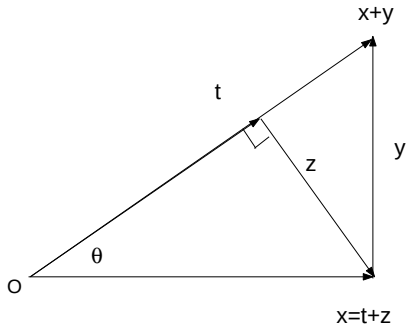
ALGORITHM : Generic Block Davidson
Choose an initial orthonormal matrix $V_1 = [v_1, \cdots, v_s] \in \mathbb{R}^{n\times s}$
Repeat
  for $k = 1 : k\max/s$,
    compute $W_k = AV_k$ ;
    compute the interaction matrix $H_k = {V_k}^T AV_k$;
    compute the $s$ smallest eigenpairs $(\lambda_{k,i}, y_{k,i})_{1\leq i\leq s}$ of $H_k$ ;
    compute the Ritz vectors $x_{k,i} = V_k y_{k,i}$ for $i = 1, \cdots, s$ ;
    compute the residuals $r_{k,i} = W_k y_{k,i} - \lambda_{k,i} x_{k,i}$ for $i = 1, \cdots, s$ ;
    if convergence then exit ;
    **compute the new directions** $(t_{k,i})_{1\leq i\leq s}$ **;** (Davidson correction)
    $V_{k+1} = \mathrm{MGS}(V_k, t_{k,1}, \cdots, t_{k,s})$ ;
  end ;
  until convergence ;
  $V_1 = \mathrm{MGS}(x_{k,1}, \cdots, x_{k,s}, t_{k,1}, \cdots, t_{k,s})$ ;
end repeat ;

### 3.6.2 How do the Davidson methods differ ?

To introduce the correction vectors $(t_{k,i})_{1\leq i\leq s}$, we assume that a known normalized vector $x$ approximates an unknown eigenvector $x + y$ of $A$, where $y$ is chosen orthogonal to $x$. The quantity $\lambda = \rho(x)$ (where $\rho(x) = x^T Ax$ denotes the Rayleigh quotient of $x$) approximates the eigenvalue $\lambda + \delta$ which corresponds to the eigenvector $x + y$ : $\lambda + \delta = \rho(x + y) = \frac{(x+y)^T A(x+y)}{\|x+y\|^2}$. The quality of the approximation is measured by the norm of the residual $r = Ax - \lambda x$. Since $r = (I - xx^T)Ax$, the residual is orthogonal to vector $x$. Let us denote $\theta$ the angle $\angle(x, x + y)$ ; let $t$ be the orthogonal projection of $x$ onto $x + y$ and $z = t - x$.

**Lemma 3.1** *The norms of the involved vectors are :*

$$\|x + y\| = \frac{1}{\cos^2\theta}, \quad (65)$$
$$\|y\| = \tan\theta, \quad (66)$$
$$\|z\| = \sin\theta, \quad (67)$$
$$\|t\| = \cos\theta. \quad (68)$$




**Proof.** Obvious. ◇

**Proposition 3.1** *With the previous notations, the correction $\delta$ to the approximation $\lambda$ of an eigenvalue, and the orthogonal correction $y$ of the corresponding approximate eigenvector $x$, satisfy:*

$$\begin{cases} (A - \lambda I)y = -r + \delta(x + y), \\ y \perp x, \end{cases} \quad (69)$$

*in which the following bounds hold :*

$$|\delta| \leq 2\|A\| \tan^2\theta, \tag{70}$$

$$\|r\| \leq 2\|A\| \tan\theta \left(\frac{\sin\theta}{\cos^2\theta} + 1\right). \tag{71}$$

**Proof.** Equation (69) is directly obtained from

$$A(x+y) = (\lambda+\delta)(x+y).$$

Moreover

$$\begin{aligned} \lambda+\delta &= \rho(t), \\ &= (1+\tan^2\theta)\ t^T A t, \\ &= (1+\tan^2\theta)(x-z)^T A(x-z), \\ &= (1+\tan^2\theta)(\lambda - 2x^T Az + z^T Az). \end{aligned}$$

Since $z$ is orthogonal to the eigenvector $t$, we obtain $x^T Az = (t+z)^T Az = z^T Az$ and therefore

$$\begin{aligned} \delta &= -z^T Az + \tan^2\theta(\lambda - z^T Az), \\ &= \tan^2\theta(\lambda - \rho(z)), or \\ |\delta| &\leq 2\tan^2\theta\ \|A\|, \end{aligned}$$

which proves (70). Bound (71) is a straight consequence of relations (69), (66) and (70). ◇

Thus the various version of the Davidson method are based on the following: at iteration $k$, the Ritz pair $(\lambda, x)$ under consideration is computed and the determined correction $y$ is added to the space $\mathcal{V}_k$ to obtain $\mathcal{V}_{k+1}$.

Solving the system (69) is not easy to solve. Since the norm of the non linear term $\delta(x+y)$ is $O(\theta^2)$ whereas $\|r\| = O(\theta)$, one may instead consider the approximate problem

$$\begin{cases} (A-\lambda I)y = -r, \\ y \perp x, \end{cases} \tag{72}$$

which has no solution except the trivial solution when $\lambda$ is an eigenvalue. Two alternatives may be considered to define another approximate problem.

The first one, which corresponds to the original Davidson's method, is to approximately solve the problem :

$$(A-\lambda I)y = -r, \tag{73}$$

and then orthogonalize $y$ with respect to the subspace $\mathcal{V}_k$. The solution cannot be the exact one else it would provide $y = -x$ and no new direction would be incorporated to yield $\mathcal{V}_{k+1}$.

The second approach is that of trace minimization and the Jacobi-Davidson schemes in which one projects the linear system (72) onto the hyperplane $(x)^\perp$ orthogonal to $x$.

More precisely, the different approaches for the correction are :

**Davidson with a preconditionner :** solve

$$(M - \lambda I)y = -r, \tag{74}$$

where $M$ is some preconditionner of $A$. In the original Davidson's method, $M$ is taken as the diagonal of $A$.

**Davidson and Inverse Iteration :** when $\lambda$ is close to an eigenvalue, solve approximately

$$(A - \lambda I)y = -r, \tag{75}$$

by an iterative method with a fixed number of iterations. In such situation, the method is similar to inverse iteration in which the ill-conditioning of the system provokes an error which is in the direction of the sought after eigenvector. The iterative solver, however, must be adapted to symmetric indefinite systems.

**Trace Minimization and Jacobi-Davidson schemes:** solve the problem :

$$\begin{cases} Q(x)(A - \lambda I)y = -r, \\ y \perp x, \end{cases} \tag{76}$$

where $Q(x) = I - xx^T$ is the orthogonal projection with respect to $x$. The system is solved iteratively. As in the previous situation, the iterative solver must be adapted for symmetric indefinite systems (the system matrix can be expressed as $Q(x)(A - \lambda I)Q(x)$).

A recent study by Simoncini and Elden [85] compares the Rayleigh Quotient method, correction via "Davidson and Inverse Iteration", and the Newton Grassmann method which corresponds to corrections via Trace Minimization or Jacobi-Davidson. The study concludes that the two correction schemes have comparable behavior. [85] also provides a stopping criterion for controlling the inner iterations of an iterative solver for the correction vectors.

### 3.6.3 Application to the computation of the smallest singular value

The smallest singular value of a matrix $A$ may be obtained by applying one of the various versions of the Davidson methods to obtain the smallest eigenvalue of the matrix $C = A^T A$, or to obtain the innermost positive eigenvalue of the 2-cyclic augmented matrix in (3). We assume that one has the basic kernels for matrix-vector multiplications, including the multiplication of the transpose of the original matrix $A$ by a vector. Multiplying the transpose of a matrix by a vector is considered a drawback, and whenever possible, the so-called "transpose-free" methods should be used. Even though one can avoid such a drawback when dealing with the interaction matrix $H_k = {V_k}^T C V_k = A{V_k}^T A V_k$, we still have to compute the residuals corresponding to the Ritz pairs which do involve multiplication of the transpose of a matrix by a vector.

For the regular single-vector Davidson method, the correction vector is obtained by approximately solving the system

$$A^T A t_k \quad = \quad r_k. \tag{77}$$

Obtaining an exact solution of (77) would yield the Lanczos algorithm applied to $C^{-1}$. Once the Ritz value approaches the square of the sought after smallest singular value, it is recommended that we solve (77) without any shifts; the benefit is that we deal with a fixed symmetric positive definite system matrix.

The approximate solution of (77) can be obtained by performing a fixed number of iterations of the Conjugate Gradient scheme, or by solving an approximate linear system $Mt_k = r_k$ with a direct method. The latter has been theoretically studied in [72] in which $M$ is obtained from approximate factorizations :

**Incomplete LU factorization of $A$ :** Here, $M = U^{-1}L^{-1}L^{-T}U^{-T}$, where $L$ and $U$ are the products of an incomplete $LU$ factorization of $A$ obtained via the so called `ILUTH`. In such a factorization, one drops entries of the reduced matrix $A$ which are below a given threshold. This version of the Davidson method is called `DAVIDLU`.

**Incomplete QR factorization of $A$:** Here, $M = R^{-1}R^{-T}$, where $R$ is the upper triangular factor of an incomplete QR factorization of $A$. This version of the Davidson method is called `DAVIDQR`.

**Incomplete Cholesky of $A^TA$:** Here, $M = L^{-T}L^{-1}$ where $L$ is the lower triangular factor of an incomplete Cholesky factorization of the normal equations. This version of the Davidson method is called `DAVIDIC`.

Even though the construction of any of the above approximate factorizations may fail, experiments presented in [72] show the effectiveness of the above three preconditioners whenever they exist. It is also shown that `DAVIDQR` is slightly more effective than either `DAVIDLU` or `DAVIDIC`.

Similar to trace minimization, the Jacobi-Davidson method can be used directly on the matrix $A^TA$ to compute the smallest eigenvalue and the corresponding eigenvector. More recently, the Jacobi-Davidson method has been adapted in [48] for obtaining the singular values of $A$ by considering the eigenvalueproblem corresponding to the 2-cyclic augmented matrix.

## Summary

Computing the smallest singular value often involves a shift and invert strategy. The advantage of Trace Minimization method and of Davidson methods over Lanczos or subspace iteration methods is to only use an approximation of the inverse through a preconditionner or through a partial resolution of an iterative procedure. Trace Minimization and Davidson methods are all based on a Newton-like correction. With the former a fixed sized basis is updated at each step while with the latter an extension of the subspace is invoked at each iteration except when restarting. One may assume that expanding the subspace usually speeds up the convergence. Often, however, using a subspace with fixed dimension results in a more robust scheme, especially if the size of the subspace is well adapted to the eigenvalue separation (i.e. singular value separation for our problem).

# 4 Parallel computation for sparse matrices

In this section, we present approaches for the parallelization of the two most important kernels: matrix-vector products and basis orthogonalization. The experiments were done on a 56-processor distributed memory machine, an Intel Paragon XP/S i860, where communications between processors were handled by the MPI library. We conclude with the description of a parallel computation of the smallest singular value of a family of sparse matrices. In that case, the experiment was done on a cluster of workstation.

## 4.1 Parallel sparse matrix-vector multiplications

Matrix-vector multiplications are usually the most CPU-time consuming operations in most iterative solvers. As mentioned earlier, the methods devoted to SVD computations involve multiplication by the original matrix and by its transpose as well. Obviously, only one copy of the matrix is stored. We present here the approach described in [47] which considers a matrix stored using the CSR format outlined in Section 3.1.

The main goal of data partitioning is to define large-grain local tasks and to overlap communications with computations. This is not always possible as efficiency depends on the matrix structure. Because of the row-oriented compact storage of the sparse matrix, data allocation is performed by rows :

**Data allocation :**

$$A = \left[\begin{array}{c} A^0 \\ \hline A^1 \\ \hline \vdots \quad \vdots \quad \vdots \quad \vdots \\ \hline A^{p-1} \end{array}\right] \qquad v = \left[\begin{array}{c} v_0 \\ \hline v_1 \\ \hline \vdots \\ \hline v_{p-1} \end{array}\right] \qquad \begin{array}{l} \longrightarrow Node\ 0 \\ \longrightarrow Node\ 1 \\ \longrightarrow \vdots \\ \longrightarrow Node\ (p-1) \end{array}$$

On each processor, the matrix components are stored row-wise as well. These, in turn, are organized into two parts: *local* and *exterior*. On processor $k$, the local part, denoted $A_{loc,k}$ defines the component which do not require communication during the multiplication. On the same processor, the exterior part is split into the blocks $A^j_{ext,k}$ $(j \neq k)$ as shown below.

**Data partition for the multiplications by $A_z$ and $A_z{}^H$ :**

$$A = \left[\begin{array}{c|c|c|c|c} A_{loc,0} & A^1_{ext,0} & \cdots & \cdots & A^{p-1}_{ext,0} \\ \hline A^0_{ext,1} & A_{loc,1} & \cdots & \cdots & A^{p-1}_{ext,1} \\ \hline \vdots & & & \vdots & \vdots \\ \hline \vdots & & & \vdots & \vdots \\ \hline A^0_{ext,p-1} & A^1_{ext,p-1} & \cdots & \cdots & A_{loc,p-1} \end{array}\right] \qquad v = \left[\begin{array}{c} v^0_{loc} \\ \hline v^1_{loc} \\ \hline \vdots \\ \hline \vdots \\ \hline v^{p-1}_{loc} \end{array}\right] \qquad \begin{array}{l} \longrightarrow Node\ 0 \\ \longrightarrow Node\ 1 \\ \longrightarrow \vdots \\ \longrightarrow \vdots \\ \longrightarrow Node\ (p-1) \end{array}$$

**Algorithms for the matrix-vector multiplication**

$k$ = processor number ; $p$ = number of processors.

**Matrix-vector multiplication** $v \to Av$ **:**

**Step 1:** Send the needed components of $v_{loc}^k$ to other processors.

**Step 2:** Compute $w^k = A_{loc,k} v_{loc}^k$.

**Step 3:** Receive the needed components of $v_{loc}^j$ for $j \neq k$ from other processors.

**Step 4:** Compute $w^k = w^k + \sum_{j \neq k} A_{ext,k}^j v_{loc}^j$.

**Matrix transpose-vector multiplication** $v \to A^T v$ **:**

**Step 1:** Compute $y^j = A_{ext}^{jT} v_{loc,j}^k$ for $j \neq k$.

**Step 2:** Send $y^j$ for $j \neq k$ to node number $j$.

**Step 3:** Compute $w^k = A_{loc,k}^T v_{loc}^k$.

**Step 4:** Receive $y^{j'}$ with $j' \neq k$ from other processors.

**step 5:** Compute $w^k = w^k + \sum_{j' \neq k} y^{j'}$.

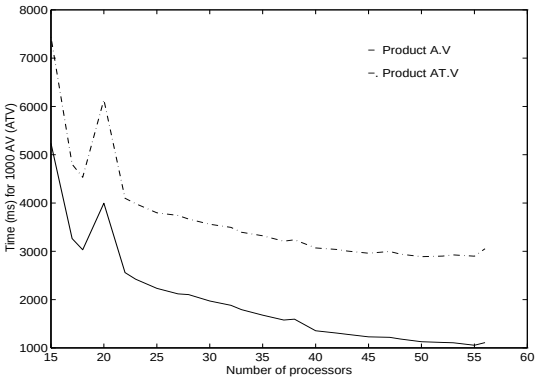
Matrix DW8192 of the Matrix Market [59] test suite was used to measure the performance of the above two matrix-vector multiplication schemes. Run times and speed-ups are displayed in Table 8, showing that one should avoid matrix-vector multiplications involving the transpose of the stored matrix, at least on architectures similar to the Intel Paragon.

## 4.2 A parallel scheme for basis orthogonalization

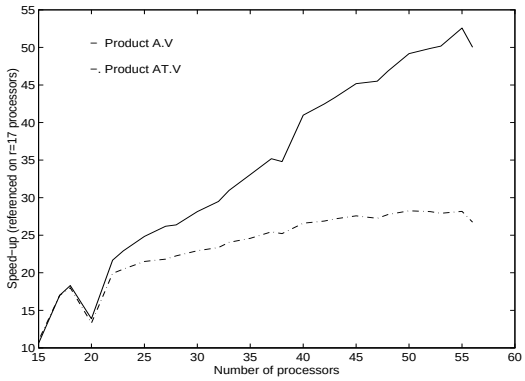
One of the crucial procedures which must be optimized for maximum performance, is the construction of an orthogonal basis. Let us assume that at some stage, the factor $Q$ of the QR factorization of a matrix $A \in \mathbb{R}^{m \times n}$ $(m \gg n)$ must be computed on a parallel computer with distributed memory.

The most common procedure, that of the Modified Gram Schmidt algorithm, does not offer effective parallelism for the case at hand $(m \gg n)$. We present here a better procedure from a previous study, e.g. see [83], in which the QR factorization of $A$ is obtained into two steps : `ROCDEC` which computes the triangular factor $R$ by combining Householder and Givens transformations, and `ROCVEC` which accumulates the orthogonal transformations to compute vector $Qy$ for any vector $y$. Assuming that the columns of $A$ are distributed on a ring of $r$ processors, see Table 9, the two procedures are given below.

$n = 8192,\ n_z = 41746$




CPU Time for 1000 $Av/A^Tv$




Speed-up for $Av/A^Tv$

Table 8: Timings for matrix-vector multiplications

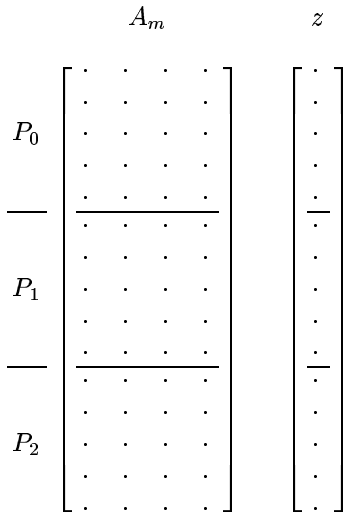



Table 9: Distribution by contiguous rows.

ALGORITHM : `ROCDEC`

initialize $\left(N, m, N_{\text{loc}}, A_{\text{loc}}(1 : N_{\text{loc}}, :)\right)$ ;
for $j := 1 : m$ do
    create my `reflector`[j] and update my local columns
    if `myid()` $= 0$ then
        send row $A_{\text{loc}}(j, j : m)$ to `myright()`
    else
        receive row(j:m) from `myleft()`
        create my `rotation`[1,j] to annihilate $A_{\text{loc}}(1, j)$
        if `myid()` $\neq r - 1$ update and send row(j:m) to `myright()`
    endif
endfor

At completion the factor Q is explicitly available but is shared in the same fashion that the original matrix $A_m$ at the beginning. Hence, the product $z = Qy$ is trivial.

```
ALGORITHM : ROCVEC
k := myid() ;
z_loc := 0 ;
{ apply orthogonal transformations in reverse order }
for j := m : 1 step −1 do
    if (k = 0) then
        receive z_loc(j) from myright()
        i := j ;
    else
        if (k = r − 1) then
            apply my rotation[1, j] on [y(j), z_loc(1)]
            send the updated y(j) to myleft()
        else
            receive yj from myright()
            apply my rotation[1, j] on [yj, z_loc(1)]
            send the updated yj to myleft()
        endif
        i := 1 ;
    endif
    apply my reflector[j] on z_loc(i : N_loc)
endfor
```

**Data distribution.** The factor R is sent to processor $P_{r-1}$. To compute the product $z = Qy$ the vector $y \in \mathbb{R}^m$ must be at the beginning in processor $P_{r-1}$. The resulting vector $z \in \mathbb{R}^N$ is split at completion.

**Parallelism.** The application of Householder reflectors is a completely independent stage. However, to annihilate the remaining nonzero elements via Givens rotations, it is necessary to transport a row portion along $r$ processors. But as soon as that portion is passed through a processor, the latter applies its next reflector so that there is an overlapping of its computations and the transfer of that row portion to the other processors. Since there will be a total of $m$ transfers of this sort during the whole factorization, communication overheads will be masked if $m \gg r$. In summary, we expect a reasonable speed-up when $\frac{n}{r} \gg m$ and $m \gg r$.

**Experiment.** The following experiment illustrates the superior scalability of the scheme ROC (ROC=ROCDEC+ROCV with respect to the Modified Gram-Schmidt procedure (MGS). The goal is to compute $Qy$ where $Q$ is the Q-factor in the QR-factorization of $A$. In Table 10, we give the running times (in seconds) of the two algorithms with a constant computing load per processor. The column length is $n = 10^4 r$ and the number of columns is $m = 40$. The algorithm ROC appears to be much more scalable than MGS.

| $r$ | MGS | ROC |
|---|---|---|
| 1 | 0.16E+01 | 0.18E+01 |
| 2 | 0.21E+01 | 0.19E+01 |
| 4 | 0.25E+01 | 0.20E+01 |
| 8 | 0.28E+01 | 0.20E+01 |
| 16 | 0.32E+01 | 0.21E+01 |
| 24 | 0.34E+01 | 0.21E+01 |
| 32 | 0.36E+01 | 0.22E+01 |
| 56 | 0.40E+01 | 0.24E+01 |

Table 10: Scalability of the orthogonalization process (as illustrated by the running time in seconds)

## 4.3 Computing the smallest singular value on several processors

As mentioned earlier (see Proposition 1.1) the smallest singular value of $A$ can be computed from the largest eigenvalue of the matrix

$$B = \begin{pmatrix} 0 & R^{-1} \\ R^{-T} & 0 \end{pmatrix}, \tag{78}$$

where $A = QR$ is a $QR$ decomposition of $A$. Typically, one would couple a Lanczos algorithm to a $QR$ decomposition to compute the largest eigenvalue of $B$ and hence compute the smallest singular value of $A$.

For large matrix dimensions, the matrix $R$ can be too large to fit in the fast memory. Furthermore, each iteration of the Lanczos algorithm requires the solutions of two linear systems based on $R$ and $R^T$. Therefore, parallel $QR$ decomposition algorithms and parallel system solvers are required to efficiently compute the smallest singular value.

The multi-frontal $QR$ decomposition (MFQRD), presented in [73], allows a large granularity for parallelism. The MFQRD starts by building a dependency graph that expresses the connections between successive steps involved in the Householder reflections. The tree nodes in the elimination tree correspond to matrix columns where column $a$ is a parent of column $b$ if the application of the Householder reflection corresponding to column $b$ alters column $a$. The computation of the Householder reflections are totally independent for separated tree leafs. Therefore, the reduction of independent subtrees can be done in parallel. On a cluster of workstations, one processor computes the elimination tree, isolates a set of independent subtrees and scatters the subtrees among the different processors of the cluster. Each processor retrieves a subtree and operates the Householder reflections. This process leads to a scattered $R$ matrix (row wise) adequate for parallel solvers. For a detailed description of this procedure, the reader is refered to [61].

Table 11 shows the wall-clock time needed to compute $\sigma_{\min}$ for three test matrices $S_1 \in \mathbb{R}^{1890\times 1890}$, $S_2 \in \mathbb{R}^{3906\times 3906}$ and $S_3 \in \mathbb{R}^{32130\times 32130}$. Although the test matrices are obtained from actual applications, we do not discuss here the physical interpretation of the results. It can be observed that parallelizing the computation of $\sigma_{\min}$ is not beneficial for $S_1$ and $S_2$, but for $S_3$ speedups of 1.66 and 2.29 are achieved for the $QR$ decomposition and the Lanczos algorithm, respectively. The total speedup for obtaining $\sigma_{\min}$ is 1.85 with a corresponding efficiency is 61%.

| Multi-frontal $QR$ decomposition | | | | |
|---|---|---|---|---|
| Matrix | 1 Proc. | 3 Procs. | Speedup | Efficiency |
| $S_1$ | 3.7 | 3.6 | 1.02 | 0.34 |
| $S_2$ | 41.8 | 30.8 | 1.35 | 0.45 |
| $S_3$ | 2353.0 | 1410.0 | 1.66 | 0.55 |
| Lanczos algorithm | | | | |
| Matrix | 1 Proc. | 3 Procs. | Speedup | Efficiency |
| $S_1$ | 0.7 | 5.2 | 0.13 | 0.04 |
| $S_2$ | 2.1 | 15.6 | 0.14 | 0.05 |
| $S_3$ | 1479.0 | 647.0 | 2.29 | 0.76 |
| $\sigma_{min}$ | | | | |
| Matrix | 1 Proc. | 3 Procs. | Speedup | Efficiency |
| $S_1$ | 4.4 | 8.8 | 0.49 | 0.16 |
| $S_2$ | 43.9 | 46.5 | 0.94 | 0.32 |
| $S_3$ | 3822.0 | 2057.0 | 1.85 | 0.61 |

Table 11: Parallel computation of $\sigma_{\min}$ on a small cluster of machines.

# 5 Application : parallel computation of a pseudo-spectrum

As mentioned earlier in Section 1.3, the computation of the pseudo-spectrum of a matrix $A$ involves a large volume of arithmetic operations. It is now commonly accepted that path following algorithms which compute the level curve

$$\Gamma_\epsilon = \{z \in \mathbb{C} \mid \sigma_{\min}(A - zI) = \epsilon\}, \tag{79}$$

are of much less complexity than methods based on grid discretization[92]. The first attempt in this direction was published by Brül [15]. Based on a continuation with a predictor-corrector scheme, the process may fail in the case of angular discontinuities along the level curve [8, 55]. In [90], Wright and Trefethen use the upper Hessenberg matrix constructed after successive iterations of the implicitly restarted Arnoldi algorithm to cheaply compute an approximation of the pseudo-spectrum. However, they show that for highly non-normal matrices the computed pseudo-spectrum is a low approximation of the exact one.

## 5.1 Parallel Path Following Algorithm using Triangles

In this section, we present the Parallel Path following Algorithm using Triangles (PPAT) for computing the level curve $\Gamma_\epsilon$. For a detailed description of this algorithm, the reader is referred to [61, 62].

PPAT uses a numerically stable algorithm that offers a guarantee of termination even in the presence of round-off errors. Furthermore, the underlying algorithm can handle singular points along the level curve of interest without difficulty.

The main idea is to line up a set of equilateral triangles along the level curve as presented in Figure 2 and use a bisection algorithm to compute a numerical approximation of the pseudo-spectrum. More specifically,

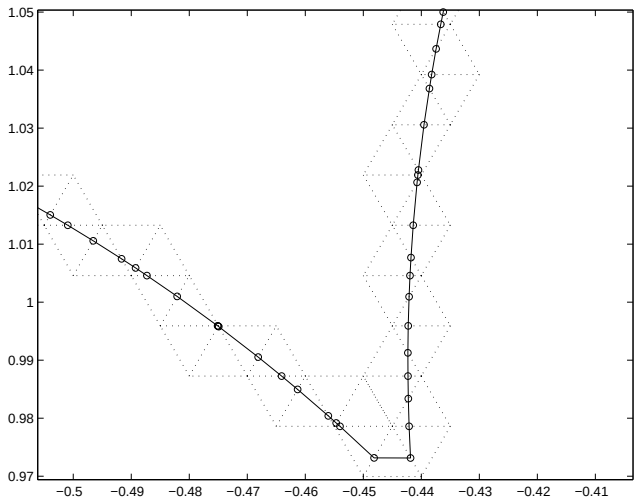



Figure 2: Computing a level curve using PPAT.

given a mesh of the complex plane with equilateral triangles, for any triangle $T_i$ of the mesh which intersects the sought level curve, an adjacent triangle $T_{i+1}$ of the same type is defined as successor of $T_i$. Therefore, from a starting triangle $T_0$, a chain of triangles $T_1, \cdots, T_n$ such that $T_n = T_0$ is defined. In [62], it is proven that to compute a level curve of length $l$, the number of equilateral triangles of size $\tau$ satisfies

$$\frac{l}{\tau} \leq n \leq \frac{10l}{\sqrt{3}\tau}. \tag{80}$$

PPAT is based on master-slave model where a master node controls a set of slave nodes capable of computing $\sigma_{\min}(A - zI)$ for a given complex value $z$ and of extracting $z \in \Gamma_\epsilon$ from a segment $[z_l, z_g]$, assuming $\sigma_{\min}(A - z_l I) \leq \epsilon < \sigma_{\min}(A - z_g I)$. For that purpose, $A$ is broadcast to all workers. Tasks are queued in a task list managed by the master. Given a triangle $T_i$ along the level curve, the master spawns two tasks; the first computes $T_{i+1}$ while the second extracts a new point of the level curve. The dynamic task scheduling allows better load balancing among the different processors of a heterogeneous network of workstations.

PPAT exploits the fact that multiple level curve slices can be computed simultaneously. The main idea is to locate different starting triangles along the level curve and use each triangle to compute a level curve slice. To get through successfully, the triangles of the different computed slices should align perfectly. Therefore, a prefixed lattice is used for all level curve slices. Furthermore, PPAT proceeds in both directions on a single level curve slice to achieve higher speedups.

## 5.2 Speedup and Efficiency

It is shown in [61] that if $n$ is the number of equilateral triangles built to compute an approximation of a given level curve using a single slice, $p$ the number of processors and $q$ the number of evaluations of $\sigma_{\min}$ for each bisection process, then the speedup is given by

$$S_p = \begin{cases} \frac{2n(q+1)}{n+2q} & \text{if } p \geq 2q+2, \\ \frac{2np(q+1)}{p^2-2p+2n(q+1)} & \text{if } p < 2q+2. \end{cases} \tag{81}$$

For large values of $n$, the speedup and efficiency can be expressed as

$$S_p = \min(p, 2q+2) + O(1/n),$$

and

$$E_p = \min\left(1, \frac{2q+2}{p}\right) + O(1/n).$$

The upper bound of the speedup is given by $S_{\max} = 2q+2$. This limit is theoretically achieved whenever $p \geq 2q+2$.

## 5.3 Test problems

Three test matrices were selected from the Matrix Market suite [59]. Table 12 summarises their characteristics ($Nz$ is the number of non zero entries and $t_{\sigma_{\min}}$ is the average computation time for $\sigma_{\min}(A - zI)$). The application uses up to 20 workers, where the master process shares the same physical processor as the first worker. The underlying hardware is a low cost general purpose network of personal computers (Pentium III, 600MHz, 128MB RAM).

| Matrix | Order | Nz | $\|\|A\|\|_F$ | $t_{\sigma_{\min}}$ |
|---|---|---|---|---|
| Olm1000 | 1000 | 3996 | $1.3 \times 10^6$ | 0.27s |
| Rdb3200L | 3200 | 18880 | $2.8 \times 10^3$ | 15.14s |
| Dw8192 | 8192 | 41746 | $1.6 \times 10^3$ | 114.36s |

Table 12: Test matrices from the non-Hermitian eigenvalueproblem (NEP) collection.

Figure 3 displays speedups and efficiencies for computing 100 points on a given level curve with PPAT. The best performance was observed for matrix `Dw8192` which is the largest of the set; a speedup of 10.85 using 12 processors which corresponds to a 90% efficiency. The lowest efficiency obtained was 63% for `Olm1000` on 13 processors. In a more realistic approach, we have used the 70 processors of the PARASKI cluster at IRISA to compute, in 41 seconds, the level curve $\epsilon = 0.05$ of `Olm1000` split into 15 slices. A single processor required 4020 seconds to compute the same level curve. The corresponding speedup is 98 with an efficiency greater than 1, indicating higher data locality realized for each processor. This result indicates the favorable scalability of PPAT since PARASKI is a heterogeneous cluster of processors ranging from PII to PIV ; the sequential time is measured on a randomly selected PIII of the cluster.

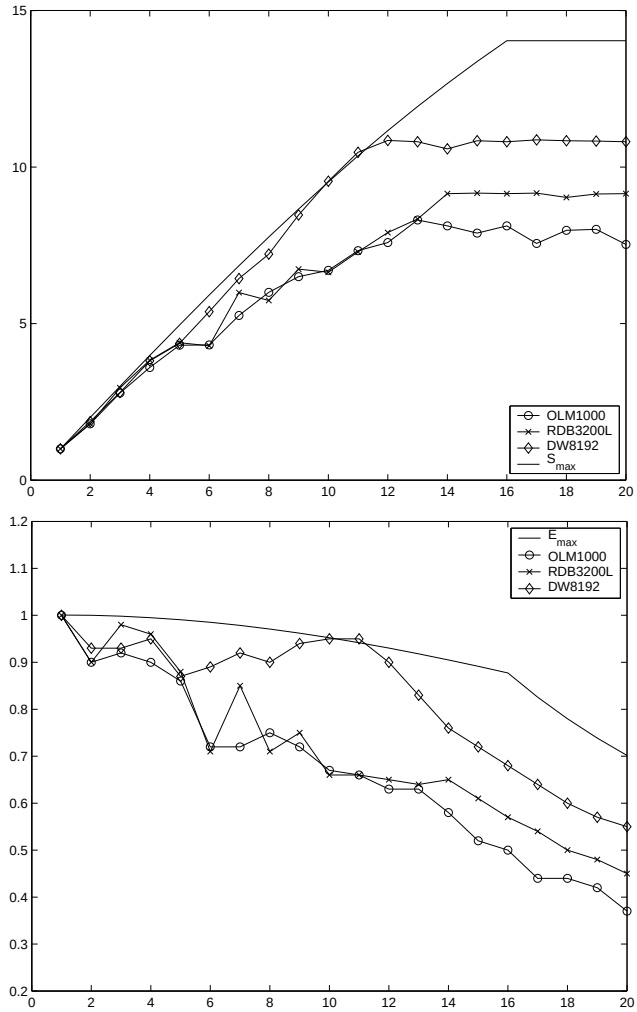



Figure 3: Speedup and Efficiency of PPAT.

# References


[1] J.O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.

[2] P. Amestoy, I. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numerical Linear Algebra with Applications*, 3:275–300, 1996.

[3] P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Meth. in Appl. Mech. and Engin.*, 184(2-4):501–520, 2000.

[4] P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, J. Koster, and M. Tuma. Mumps 4.1.6. http://www.enseeiht.fr/lima/apo/MUMPS/, March 2000.

[5] S.F. Ashby, T.A. Manteuffel, and P.E. Saylor. A taxonomy for Conjugate Gradient Methods. *SIAM Journal on Numerical Analysis*, 26:1542–1568, 1990.

[6] R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods - 2nd edition.* SIAM / netlib, Philadelphia, PA, 1994.

[7] F.L. Bauer. Das verfahred der treppeniteration und verwandte verfahren zur losung algebraischer eigenwertprobleme. *ZAMP*, 8:214–235, 1957.

[8] C. Bekas and E. Gallopoulos. Cobra: Parallel path following for computing the matrix pseudospectrum. *Parallel Computing*, 27(14):1879–1896, 2001.

[9] M. Berry, B. Parlett, and A. Sameh. Computing extremal singular triplets of sparse matrices on a shared-memory multiprocessor. *Int. Journ. High Speed Comput.*, 2:239–275, 1994.

[10] M.W. Berry. Large scale singular value computations. *Int'l. J. of Supercomputer Applications*, 6:13–49, 1992.

[11] M.W Berry and A.H Sameh. An overview of parallel algorithms for the singular value and symmetric eigenvalue problems. *Computational and Applied Mathematics*, 27:191–213, 1989.

[12] I. Brainman and S. Toledo. Nested-dissection orderings for sparse lu with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 23(4):998–1012, 2002.

[13] R.P. Brent and F.T. Luk. The solution of singular value and symmetric eigenproblems on multiprocessor arrays. *SIAM J. Sci. Statist.*, 6:69–84, 1985.

[14] R.P. Brent, F.T. Luk, and C. Van Loan. Computation of the singular value decomposition using mesh connected processors. *VLSI Comput. Systems*, 1(3):242–270, 1985.

[15] M. Brühl. A curve tracing algorithm for computing the pseudospectrum. *BIT*, 36(3):441–454, 1996.

[16] J.R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31:162–179, 1977.

[17] J.-P. Charlier and P. Van Dooren. On kogbetliantz's svd algorithm in the presence of clusters. *Linear Algebra Appl.*, 95:135–160, 1987.

[18] 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.

[19] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson method. *SIAM, Journal on Scientific and Statistical Computing*, 15:1:62–76, 1994.

[20] J.K. Cullum and R.A. Willoughby. *Lanczos Algorithm for Large Symmetric Eigenvalue Computations*, volume 1. Birkhauser, 1985.

[21] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Comput. Phys.*, 17:87–94, 1975.

[22] T. A. Davis. Umfpack version 4.0. http://www.cise.ufl.edu/research/sparse/umfpack, April 2002.

[23] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Software*, 25:1–19, 1999.

[24] J. Demmel, Ming Gu, S. Eisenstat, and Z. Drmač I. Slapničar, K. Veselić. Computing the singular value decomposition with high accuracy. *Linear Algebra Appl.*, 299(1-3):21–80, 1999.

[25] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Joseph.

[26] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Trans. on Math. Software*, 14(1):1–17, 1988.

[27] J. Dongarra and D. Sorensen. A fast algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8(2):s139–s154, 1987.

[28] J. J. Dongarra, I. S. Duff, D. S. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, 1991.

[29] I. Duff, A. Erisman, and J Reid. *Direct methods for sparse matrices.* Oxford Science Publications, 1992.

[30] I. S. Duff. A review of frontal methods for solving linear systems. *Comput. Physics Comm.*, 97(1-2):45–52, 1996.

[31] G. Forsythe and P. Henrici. The cyclic jacobi method for computing the principal values of a complex matrix. *Trans. Amer. Math. Soc.*, 94:1–23, 1960.

[32] R. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM journal on scientific computing*, 14:470–482, 1993.

[33] R. Freund and N. Nachtigal. QMR : a quasi-minimal residual method for non-Hermitian linear systems. *Numerische mathematik*, 60:315–339, 1991.

[34] K. Gallivan, W. Jalby, and U. Meier. The use of blas3 in linear algebra on a parallel processor with a hierarchical memory. *SIAM J. Sci. Stat. Comput.*, 18(6):1079–1084, 1987.

[35] J. R. Gilbert, X. S. Li, E. G. Ng, and B. W. Peyton. Computing row and column counts for sparse qr and lu factorization. *BIT*, 41(4):693–710, 2001.

[36] S. K. Godunov. Spectral portraits of matrices and criteria of spectrum dichotomy. In L. Atanassova and J. Hezberger, editors, *3rd International IMACS-CAMM Symposium on Computer Arithmetic and Enclosure Methods*, Amsterdam, 1992.

[37] G.H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal.,*, 2(3):205–224, 1965.

[38] G.H. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins, second edition edition, 1989.

[39] G.H. Golub, F.T. Luk, and M.L. Overton. A block lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Trans. on Math. Software*, 7(2):149–169, 1981.

[40] G.H. Golub and C. Reinsch. *Singular Value Decomposition and Least Squares Solutions.* Springer-Verlag, 1971.

[41] G.H. Golub and R.R. Underwood. *The Block Lanczos Method for Computing Eigenvalues*, pages 361–377. Academic Press, 1977.

[42] G.H Golub and C.F Van Loan. *Matrix Computations. third edition.* John Hopkins, 1996.

[43] J. Grcar. Analysis of the lanczos algorithm and of the approximation problem in richardson's method. Technical report, The Univ. of Illinois, 1981. Ph.D. Thesis.

[44] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences.* Springer Verlag, 1994.

[45] P. Henrici. On the speed of convergence of cyclic and quasicyclic jacobi methods for computing eigenvalues of hermitian matrices. *Soc. Indust. Appl. Math.*, 6:144–162, 1958.

[46] M.R. Hestens. Inversion of matrices by biorthogonalization and related results. *Soc. Indust. Appl. Math.*, 6:51–90, 1958.

[47] V. Heuveline, B. Philippe, and M. Sadkane. Parallel computation of spectral portrait of large matrices by davidson type methods. *Numer. Algorithms*, 16(1):55–75, 1997.

[48] M. E. Hochstenbach. A jacobi-davidson type svd method. *SIAM J. Sci. Compu.*, 23(2):606–628, 2001.

[49] H. Hoteit, J. Erhel, R. Mosé, B. Philippe, and P. Ackerer. Numerical reliability for mixed methods applied to flow problems in porous media. *J. Comput. Geosciences*, 6:161–194, 2002.

[50] W. Jalby and B. Philippe. Stability analysis and improvement of the block gram-schmidt algorithm. *SIAM J. Sci. Stat. Comput.*, 12(5):1058–1073, 1991.

[51] H.F. Kaiser. The jk method: A procedure for finding the eigenvectors and eigenvalues of a real symmetric matrix. *Comput. J.*, 15(33):271–273, 1972.

[52] X. S. Li. Superlu version 2. http://www.nersc.gov/ xiaoye/SuperLU/, September 1999.

[53] S. Lo, B. Philippe, and A.H. Sameh. A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8(2):s155–s165, 1987.

[54] D.G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, 1973.

[55] S.H. Lui. Computation of pseudospectra by continuation. *SIAM J. Sci. Comput*, 28(2):565–573, 1997.

[56] F. Luk and H. Park. A proof of convergence for two parallel jacobi svd algorithms. *IEEE Trans. Comput.* (to appear).

[57] F. Luk and H. Park. On the equivalence and convergence of parallel jacobi svd algorithms. *IEEE Trans. Comput.*, 38(6):806–811, 1989.

[58] F.T. Luk. Computing the singular value decomposition on the illiac iv. *ACM Trans. Math. Soft.*, 6(4):524–539, 1980.

[59] Matrix Market. http://math.nist.gov/MatrixMarket/.

[60] G. Meurant. *Computer solution of large linear systems.* North Holland, Amsterdam, 1999.

[61] D. Mezher and B. Philippe. Parallel computation of pseudospectra of large sparse matrices. *Parallel Computing*, 28(2):199–221, 2002.

[62] D. Mezher and B. Philippe. Pat - a reliable path-following algorithm. *Numerical Algorithms*, 29:131–152, 2002.

[63] R. B. Morgan and D. S. Scott. Generalizations of davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Statist. Comput.*, 7:817–825, 1986.

[64] J.C. Nash. A one-sided transformation method for the singular value decomposition and algebraic eigenproblems. *Comput. J.*, 18(1):74–76, 1975.

[65] C. Paige and P. Van Dooren. On the quadratic convergence of kogbetliantz's algorithm for computing the singular value decomposition. *Linear Algebra Appl.*, 77:301–313, 1986.

[66] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM journal on numerical analysis*, 12:617–629, 1975.

[67] C.C. Paige. Error analysis of the lanczos algorithms for tridiagonalizing a symmetric matrix. *Inst. Math. Appl.*, 18:341–349, 1976.

[68] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.

[69] B.N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice Hall, 1980.

[70] B.N. Parlett and J.K. Reid. On the solution of a system of linear equations whose system is symmetric but not definite. *BIT*, 10:386–397, 1970.

[71] B.N. Parlett and D.S. Scott. The lanczos algorithm with selective reorthogonalization. *Math. Comp.*, 33:217–238, 1979.

[72] B. Philippe and M. Sadkane. Computation of the fundamental singular subspace of a large matrix. *Linear Algebra and its Applications*, 257:77–104, 1997.

[73] I.S. Duff P.R. Amestoy and C. Puglisi. Multifrontal qr factorization in a multiprocessor environment. *Int. J. Numer. Lin. Alg. Applt.*, (3):275–300, 1996.

[74] H. Rutishauser. Computational aspects of f.l. bauer's simultaneous iteration method. *Numer. Math.*, 13:4–13, 1969.

[75] H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numer. Math.*, 16:205–223, 1970.

[76] Y. Saad. *Iterative methods for sparse linear systems.* PWS Publishing Company, 1996.

[77] Y Saad and H Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[78] A. Sameh. On jacobi and jacobi-like algorithms for a parallel computer. *Math. Comp.*, 25:579–590, 1971.

[79] A. Sameh. *Solving the Linear Least Squares Problem on a Linear Array of Processors*, pages 191–200. Academic Press, New York, 1985.

[80] A. Sameh and Z. Tong. The trace minimization method for the symmetric generalized eigenvalue problem. *J. of Computational and Applied Math.*, 123:155–175, 2000.

[81] A.H. Sameh and J.A. Wisniewski. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM J. Numer. Anal.*, 19(6):1243–1259, 1982.

[82] A. Schonhage. Zur konvergenz des jacobi-verfahrens. *Numer. Math.*, 3:374–380, 1961.

[83] R. Sidje and B. Philippe. Parallel Krylov subspace basis computation. In J. Tankoano, editor, *CARI'94 Proceedings, Ouagadougou*, pages 421–440. INRIA - ORSTOM, ORSTOM Editions, Paris, 1994.

[84] H. Simon. Analysis of the symmetric lanczos algorithm with reorthogonalization methods. *Linear Algebra and Its Applications*, 61:101–131, 1984.

[85] V. Simoncini and L. Eldén. Inexact rayleigh quotient-type methods for eigenvalue computations. *BIT*, 42(1):159–182, 2002.

[86] G. L. G. Sleipen and H. A. van der Vorst. A jacobi-davidson iteration method for linear eigenvalue problems. *SIAM J. Mat. Anal. and Appl.*, 17(2):401–425, 1996. also SIGEST in SIAM Review, Vol. 42, No. 2, pp. 267-293.

[87] B. Smith, J. Boyce, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, and C. Moler. *Matrix Eigensystem Routines – EISPACK Guide.* Springer Verlag, second edition, 1976.

[88] G. W. Stewart and Ji-guang Sun. *Matrix perturbation theory.* Academic Press, 1990.

[89] J.-G. Sun. A note on simple non-zero singular values. *J. Comput. Math.*, (6):258–266, 1988.

[90] L.N. Trefethen T. Wright. Large-scale computation of pseudospectra using arpack and eigs. Technical report, Oxford University Computing Laboratory, June 2000.

[91] L. N. Trefethen. Pseudospectra of matrices. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis*, Dundee 1991, pages 234–266. Longman, 1992.

[92] L.N. Trefethen. Computation of pseudospectra. *Acta Numerica*, pages 247–295, 1999.

[93] R.R. Underwood. An iterative block lanczos method for the solution of large sparse symmetric eigenproblems. Technical report, Stanford University, 1975. Ph.D. thesis.

[94] H. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 13:631–644, 1992.

[95] J.H. Wilkinson. Note on the quadratic convergence of the cyclic jacobi process. *Numer. Math.*, 5:296–300, 1962.

[96] J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, 1965.

[97] J.H. Wilkinson. Almost diagonal matrices with multiple or close eigenvalues. *Linear Algebra Appl.*, 1:1–12, 1968.

[98] J.H. Wilkinson. *Inverse Iteration in Theory and in Practice.* Academic Press, 1972.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399