4 Parallel Algorithms for the Singular Value Decomposition

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

CONTENTS

ABSTRACT

The goal of the survey is to review 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 present those schemes that yield the complete decomposition, whereas for sparse matrices we describe 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, we conclude with the presentation of a parallel method for computing pseudospectra, which depends on computing the smallest singular values.

4.1 INTRODUCTION

4.1.1 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. The SVD is commonly used in the solution of unconstrained linear least squares problems, matrix rank estimation, and canonical correlation analysis. In computational science, it is commonly applied in domains such as information retrieval, seismic reflection tomography, and real-time signal processing [1].

In what follows, we will provide a brief survey of some parallel algorithms for obtaining the SVD for dense, and large sparse matrices. For sparse matrices, however, we focus mainly on the problem of obtaining the smallest singular triplets.

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 [2, 3].

Theorem 4.1 *[SVD]*
If $A \in \mathbb{R}^{m\times n}$ is a real matrix, then there exist orthogonal matrices

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

such that

$$\Sigma = U^{\mathrm{T}} A V = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}, \quad p = \min(m, n) \tag{4.1}$$

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

Definition 4.1 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 σ_i $(i = 1, \ldots, p)$, the vectors u_i and v_i are respectively the left and right singular vectors of A associated with σ_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 nonnegative values.

Theorem 4.2 *Let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ have the singular value decomposition*

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

Then, the symmetric matrix

$$B = A^{\mathrm{T}} A \in \mathbb{R}^{n\times n}, \tag{4.2}$$

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

$$A_{\text{aug}} = \begin{pmatrix} 0 & A \\ A^{\text{T}} & 0 \end{pmatrix} \tag{4.3}$$

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 A_{aug} *is called the augmented matrix.*

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

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

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

which can also be written as

$$\| r_i \|_2 = [(\| A\tilde{v}_i - \tilde{\sigma}_i\tilde{u}_i \|_2^2 + \| A^{\text{T}}\tilde{u}_i - \tilde{\sigma}_i\tilde{v}_i \|_2^2)^{1/2}]/ \| \tilde{u}_i \|_2^2 + \| \tilde{v}_i \|_2^2]^{1/2}. \tag{4.4}$$

The backward error [4]

$$\eta_i \quad = \quad \max\{\| A\tilde{v}_i - \tilde{\sigma}_i\tilde{u}_i \|_2, \ \| A^{\text{T}}\tilde{u}_i - \tilde{\sigma}_i\tilde{v}_i \|_2^2\}$$

may also be used as a measure of absolute accuracy. A normalizing factor can be introduced for assessing relative errors.

Alternatively, we may compute the SVD of A indirectly by the eigenpairs of either the $n \times n$ matrix $A^{\text{T}}A$ 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 $A^{\text{T}}A$, then

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

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

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

where σ_i is the ith nonzero singular value of A corresponding to the left singular vector u_i. The corresponding right singular vector, v_i, is then obtained as $v_i = (1/\sigma_i)A^{\text{T}}u_i$.

Computing the SVD of A via the eigensystems of either $A^{\text{T}}A$ 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 Ref. [5]). In fact, extremely small singular values of A (i.e., smaller than $\sqrt{\epsilon}\|A\|$, where ϵ is the machine precision parameter) may be computed as zero eigenvalues of $A^{\text{T}}A$ (or AA^{T}). Whereas the smallest and largest singular values of A are the lower and upper bounds of the spectrum of $A^{\text{T}}A$ or AA^{T}, the smallest singular values of A lie at the center of the

spectrum of A_{aug} in (4.3). For computed eigenpairs of $A^{\mathrm{T}}A$ and AA^{T}, the norms of the ith eigenpair residuals (corresponding to (4.4)) are given by

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

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 4.1.2. Difficulties in approximating the smallest singular values by any of the three equivalent symmetric eigenvalue problems will be discussed in Section 4.4.

When A is a square nonsingular matrix, it may be advantageous in certain cases to compute the singular values of A^{-1} which are $(1/\sigma_n) \geq \cdots \geq (1/\sigma_1)$. 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 4.3). Actually, the method can be extended to rectangular matrices of full rank by considering a QR-factorization:

Proposition 4.1 *Let $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ be of rank n. Let*

$$A = QR, \text{ where } Q \in \mathbb{R}^{m\times n} \text{ and } R \in \mathbb{R}^{n\times n},$$

such that $Q^{\mathrm{T}}Q = I_n$ and R 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 $(R^{-1}R^{-\mathrm{T}})$ or of $\begin{pmatrix} 0 & R^{-1} \\ R^{-\mathrm{T}} & 0 \end{pmatrix}$.

4.1.2 Sensitivity of the Smallest Singular Value

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 4.3 *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*

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

They satisfy the following bounds

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

Proof. See Ref. [3]. □

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

Proposition 4.2 *The relative condition number of the smallest singular value of a nonsingular matrix 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 behavior (i.e., backward stable).

Recently, some progress has been made [6]. 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 4.2.2).

Because the nonzero 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 4.4 *Let σ be a nonzero 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*

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

Proof. See Ref. [7]. □

The effect of a perturbation of the matrix on the singular vectors can be more significant than that on the singular values. The sensitivity of the singular vectors depend on the singular value distribution. When a simple singular value is not well separated from the rest, the corresponding left and right singular vectors are poorly determined. This is made precise by the following theorem, see Ref. [3], which we state here without proof. Let $A \in \mathbb{R}^{n\times m}$ $(n \geq m)$ have the SVD

$$U^{\mathrm{T}} A V = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix}.$$

Partition $U = (u_1\ U_2\ U_3)$ and $V = (v_1\ V_2)$ where $u_1 \in \mathbb{R}^n$, $U_2 \in \mathbb{R}^{n\times(m-1)}$, $U_3 \in \mathbb{R}^{n\times(n-m)}$, $v_1 \in \mathbb{R}^m$, and $U_2 \in \mathbb{R}^{m\times(m-1)}$. Partition conformally

$$U^{\mathrm{T}} A V = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \\ 0 & 0 \end{pmatrix}.$$

Given a perturbation $\tilde{A} = A + E$ of A, let

$$U^{\mathrm{T}} E V = \begin{pmatrix} \gamma_{11} & g_{12}{}^{\mathrm{T}} \\ g_{21} & G_{22} \\ g_{31} & G_{32} \end{pmatrix}.$$

Theorem 4.5 *Let $h = \sigma_1 g_{12} + \Sigma_2 g_{21}$. If $(\sigma_1 I - \Sigma_2)$ is nonsingular (i.e., if σ_1 is a simple singular value of A), then the matrix*

$$U^{\mathrm{T}} \tilde{A} V = \begin{pmatrix} \sigma_1 + \gamma_{11} & g_{12}{}^{\mathrm{T}} \\ g_{21} & \Sigma_2 + G_{22} \\ g_{31} & G_{32} \end{pmatrix}$$

has a right singular vector of the form

$$\begin{pmatrix} 1 \\ (\sigma_1{}^2 I - \Sigma_2{}^2)^{-1} h \end{pmatrix} + O(\|E\|^2).$$

It was remarked in Theorem 4.1 that computing the SVD of A could be obtained from the eigendecomposition of the matrix $C = A^{\mathrm{T}} A$ or of the augmented matrix $A_{\mathrm{aug}} = \begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix}$. It

is clear, however, that using C to compute the smallest singular value of A is bound to yield poorer result as the condition number of C is the square of the condition number of A_{aug}. It can be shown, however, that even with an ill-conditioned matrix A, the matrix C can be used to compute accurate singular values.

4.1.3 Distance to Singularity — Pseudospectrum

Let us consider a linear system defined by the square matrix $A \in \mathbb{R}^{n\times n}$. However, 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 4.6 $d(A,S) = \sigma_{\min}(A)$.

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 $(A - \sigma v u^{\mathrm{T}})u = 0$. Since $\|\sigma v u^{\mathrm{T}}\|_2 = \sigma$, apparently $B = A - \sigma v u^{\mathrm{T}} \in \mathcal{S}$ 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 4.3 *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 $\|A^{-1}\|_2 = 1/\sigma_{\min}(A)$. □

For instance in Ref. [8], 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 [9] or of ϵ-spectrum [10] was introduced:

Definition 4.2 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:

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid \exists \Delta \in \mathbb{C}^{n\times n} \text{such that } \|\Delta\| \leq \epsilon \text{ and } z \text{ is an eigenvalue of } (A + \Delta)\}. \tag{4.5}$$

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

Proposition 4.4 *The pseudospectrum is the set:*

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid \sigma_{\min}(A - zI) \leq \epsilon\}, \tag{4.6}$$

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 4.6. □

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 consideration 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 [11–13]. Section 4.5 describes one of these methods. For illustration, the pseudospectrum of a matrix from the Matrix Market [14] test suite is displayed in Figure 4.1, where several values of ϵ are shown.

In what follows, we present a selection of parallel algorithms for computing the SVD of dense and sparse matrices. For dense matrices, we restrict our survey to the Jacobi methods for obtaining all the singular triplets, a class of methods not contained in ScaLAPACK [15, 16]. The ScaLAPACK (or Scalable LAPACK) library includes a subset of LAPACK routines redesigned for distributed memory Multiple Instruction Multiple Data (MIMD) parallel computers. It is currently written in a Single-Program-Multiple-Data style using explicit message passing for interprocessor communication. It assumes matrices are laid out in a two-dimensional block cyclic fashion. The routines of the library achieve respectable efficiency on parallel computers and the software is considered to be robust. Some projects are under way, however, for making the use of ScaLAPACK in large-scale applications more user-friendly. Examples include, the PLAPACK project [17, 18], the OURAGAN project which is based on SCILAB [19], as well as projects based on MATLAB [20]. None of these projects, however, provide all the capabilities of ScaLAPACK.

Whereas for sparse matrices, we concentrate our exposition on those schemes that obtain the smallest singular triplets. We devote the last section to the vital primitives that help to assure the realization of high to performance on parallel computing platforms.

4.2 JACOBI METHODS FOR DENSE MATRICES

4.2.1 TWO-SIDED JACOBI SCHEME [2JAC]

Here, we consider the standard eigenvalue problem

$$Bx = \lambda x \tag{4.7}$$

where B is a real $n \times n$-dense symmetric matrix. One of the best known methods for determining all the eigenpairs of (4.7) was developed by the 19th century mathematician, Jacobi. We recall that Jacobi's sequential method reduces the matrix B to the diagonal form by an infinite sequence of plane rotations

$$B_{k+1} = U_k B_k U_k^{\mathrm{T}}, \quad k = 1, 2, ...,$$

where $B_1 \equiv B$, 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 θ_{ij}^k is determined so that $b_{ij}^{k+1} = b_{ji}^{k+1} = 0$, or

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

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

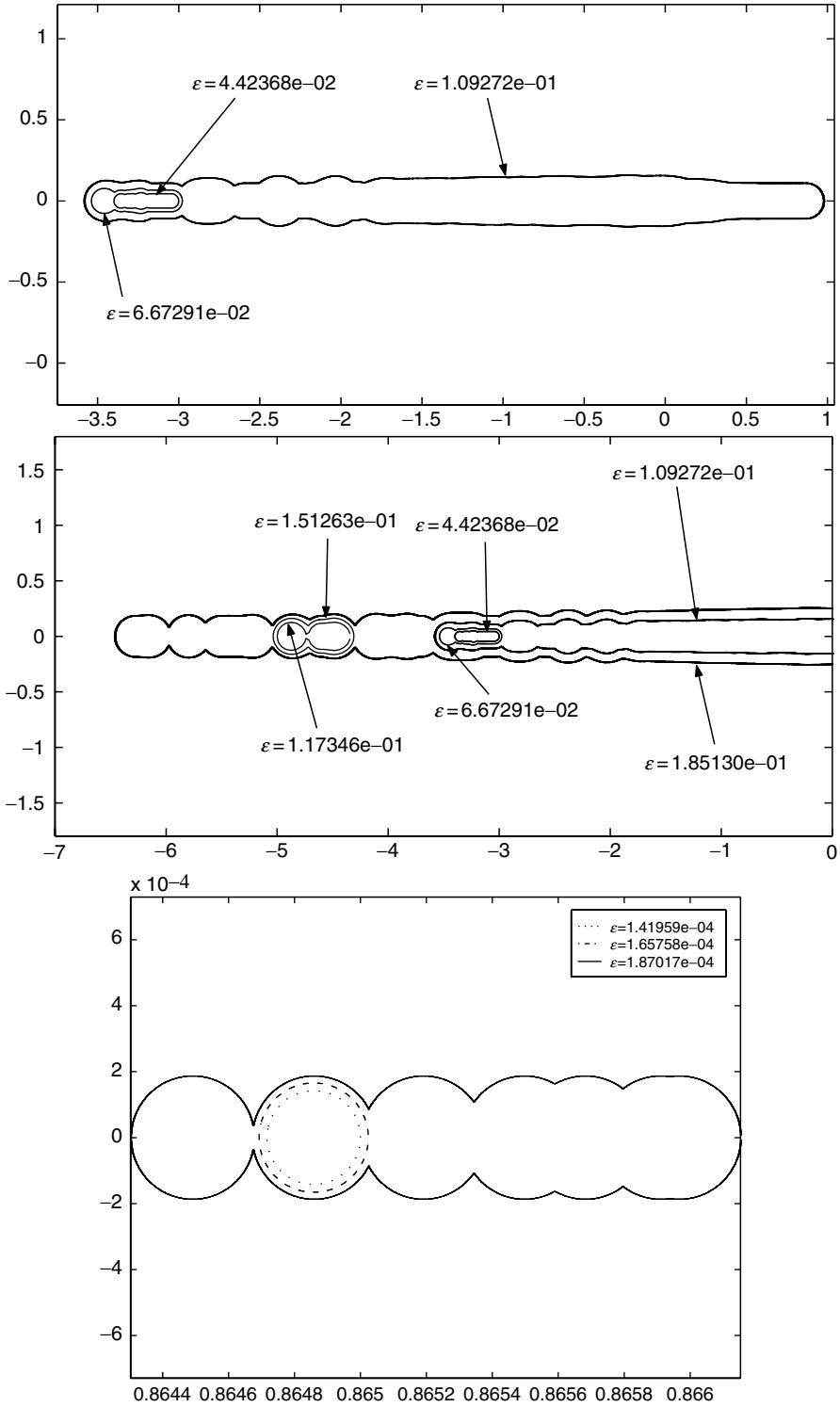


FIGURE 4.1 Portraits of the matrix DW8192.

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

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

and

$$b_{ir}^{k+1} = c_k b_{ir}^k + s_k b_{jr}^k, \tag{4.8}$$

$$b_{jr}^{k+1} = -s_k b_{ir}^k + c_k b_{jr}^k, \tag{4.9}$$

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

$$B_k = D_k + E_k + E_k^{\mathrm{T}}, \tag{4.10}$$

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

Several schemes are possible for selecting the sequence of elements b_{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 [21]. 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 Refs. [22, 23]). 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 B be of order 8 and consider the orthogonal matrix U_k as the direct sum of four 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 eight successive orthogonal transformations with each one annihilating distinct groups of four 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 $r = \left\lfloor \frac{1}{2}(n+1) \right\rfloor$, in which each transformation annihilates different $\left\lfloor \frac{1}{2}n \right\rfloor$ off-diagonal elements (see Ref. [24]). 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 [25, 26] 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

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

   for $k > 2$,

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

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

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

Step 2: (Convergence test).

1. (a) Compute $\| D_k \|_{\mathrm{F}}$ and $\| E_k \|_{\mathrm{F}}$ (see (4.10)).
   (b) If

$$\frac{\| E_k \|_{\mathrm{F}}}{\| D_k \|_{\mathrm{F}}} < \text{tolerance}, \tag{4.11}$$

   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 $\left\lfloor \frac{1}{2}n \right\rfloor$ or $\left\lfloor \frac{1}{2}(n-1) \right\rfloor$ plane rotations, depending on whether k is odd or even, respectively. The annihilation pattern for $n = 8$ is shown in table 4.1, where the integer k denotes an element annihilation via U_k.

TABLE 4.1
Annihilation Scheme for [2JAC]

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

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 (4.8) and (4.9). 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 B_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 B_k and operate with full vectors on the entirety of rows and columns i and j in (4.8) and (4.9), i.e., we are using a full matrix scheme. The product of the U_k's, which eventually yields the eigenvectors for B, is accumulated in a separate two-dimensional array by applying (4.8) and (4.9) 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 Ref. [25]), this multiprocessor algorithm can be shown to converge quadratically by following the work of Wilkinson [23] and Henrici [27].

4.2.2 One-Sided Jacobi Scheme [1JAC]

Suppose that A is a real $m \times n$-matrix with $m \gg n$ and rank $A = r$. The SVD of A can be defined as

$$A = U \Sigma V^{\mathrm{T}}, \tag{4.12}$$

where $U^{\mathrm{T}}U = V^{\mathrm{T}}V = I_n$ and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r$, $\sigma_j = 0$ for $j \geq r + 1$. 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 $A^{\mathrm{T}}A$.

As indicated in Ref. [28] for a ring of processors, using a method based on the one-sided iterative orthogonalization method of Hestenes (see also Ref. [29, 30]) is an efficient way to compute the decomposition (4.12). Luk [31] recommended this singular value decomposition scheme on the Illiac IV, and corresponding systolic algorithms associated with *two-sided* schemes have been presented in Ref. [32, 33]. We now consider a few modifications to the scheme discussed in Ref. [28] for the determination of (4.12) on shared-memory multiprocessors, e.g., see Ref. [34].

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

$$A\tilde{V} = Q = (q_1, q_2, \ldots, q_r), \tag{4.13}$$

and

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

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

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

then

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

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

$$\tilde{a}_i^{\mathrm{T}}\tilde{a}_j = 0 \quad \text{and} \quad \| \tilde{a}_i \|_2 > \| \tilde{a}_j \|_2, \tag{4.14}$$

where a_i designates the ith column of matrix A. This is accomplished by choosing

$$c = \left[\frac{\beta + \gamma}{2\gamma}\right]^{1/2} \quad \text{and} \quad s = \left[\frac{\alpha}{2\gamma c}\right], \quad \text{if} \quad \beta > 0, \tag{4.15}$$

or

$$s = \left[\frac{\gamma - \beta}{2\gamma}\right]^{1/2} \quad \text{and} \quad c = \left[\frac{\alpha}{2\gamma s}\right], \quad \text{if} \quad \beta < 0, \tag{4.16}$$

where $\alpha = 2a_i^{\mathrm{T}}a_j$, $\beta = \| a_j \|_2^2$, and $\gamma = (\alpha^2 + \beta^2)^{1/2}$. Note that (4.14) requires the columns of Q to decrease in norm from left to right, and hence the resulting σ_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 4.1, 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 4.1). 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 four 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

$$V = V_{s_1} V_{s_2} \cdots V_{s_t}, \tag{4.17}$$

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

$$\frac{(a_i^{\mathrm{T}} a_j)^2}{(a_i^{\mathrm{T}} a_i)(a_j^{\mathrm{T}} a_j)} > \text{tolerance}, \tag{4.18}$$

   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 = rank $A = k_{\min}$. *Note*: if (4.18) 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 $\sigma_i = \sqrt{(A^{\mathrm{T}}A)_{ii}}$, $i = 1, 2, \ldots, k_{\min} = r$, 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 (4.15) and (4.16), 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

$$a_i^{k+1} = c a_i^k + s a_j^k, \tag{4.19}$$

$$a_j^{k+1} = -s a_i^k + c a_j^k, \tag{4.20}$$

where a_i denotes the ith column of matrix A, and c, s are determined by either (4.15) or (4.16). 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 (4.19) and (4.20). Each processor is assigned one rotation and hence orthogonalizes one pair of the n columns of matrix A.

Following the convergence test used in Ref. [30], in Step 2, we count the number of times the quantity

$$\frac{a_i^{\mathrm{T}} a_j}{(a_i^{\mathrm{T}} a_i)(a_j^{\mathrm{T}} a_j)} \tag{4.21}$$

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 (4.13) and hence the nonzero

singular values σ_i can be obtained via the r square roots of the first r diagonal entries of $A^{\mathrm{T}}A$. The matrix $\tilde{U}$ in (4.12), 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 σ_i. Similarly, the matrix V, which contains the right singular vectors of the original matrix A, is obtained as in (4.17) 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 (4.19) and (4.20) to the $n \times n$-identity matrix. It is important to note that the use of the ratio in (4.21) is preferable over the use of $a_i^{\mathrm{T}} a_j$, since this dot-product is necessarily small for relatively small singular values.

Although our derivation of [1JAC] is concerned with the singular value decomposition of rectangular matrices, it is most effective for solving the eigenvalue problem in (4.7) for symmetric positive definite matrices. If $m = n = r$, B is a positive definite matrix, and Q in (4.13) is an orthogonal matrix. Consequently, it is not difficult to show that

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

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

4.2.3 Algorithm [QJAC]

As discussed above, [1JAC] is certainly a viable candidate for computing the SVD (4.12) 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 (4.12). In this section, we present a multiprocessor method, QJAC, which can be quite effective for computing (4.12) 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

$$A = QR, \tag{4.22}$$

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 (4.22) 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 SVD of the upper-triangular matrix R.

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

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

where $U^{\mathrm{T}}U = V^{\mathrm{T}}V = I_r$, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$, $\sigma_i > 0$ for $1 \le i \le r$, 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

$$A = QR, \tag{4.23}$$

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],

$$R = \tilde{U} \begin{bmatrix} \tilde{\Sigma} \\ 0 \end{bmatrix} V^{\mathbf{T}}, \tag{4.24}$$

where $\tilde{U}$ and V are $n \times r$-matrices having orthogonal columns ($r \equiv$ 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 $\tilde{U}$:

$$U = Q\tilde{U}, \tag{4.25}$$

where Q is the product of the Householder transformations applied in Step 1 and u_i is the ith 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 to capitalize upon the potential vectorization and parallelism inherent in [1JAC] on parallel machines with vector processors.

Charlier et al. [35] 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

$$A_{k+1} = U_k A_k V_k^{\mathrm{T}}, \quad k = 1, 2, \ldots, \tag{4.26}$$

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 σ_i is the ith singular value of A, and the products $(U_k \cdots U_2 U_1)$, $(V_k \cdots V_2 V_1)$ approach matrices whose ith column is the respective left and right singular vector corresponding to σ_i. For the case when the σ_i's are not pathologically close, Paige and Van Dooren [36] have shown that the row (or column) cyclic Kogbetliantz's method ultimately converges quadratically. For triangular matrices, Charlier and Van Dooren [37] 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_ν, where ν is the number of sweeps required for convergence. Even if we were to assume that R in (4.22) satisfies this condition for quadratic convergence of the parallel Kogbetliantz's method in [36], 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] is to determine the SVD of R lies in that the rotations defined by (4.15) or (4.16), as applied via the parallel ordering illustrated in Table 4.1, require no processor synchronization among any set of the $\left\lfloor \frac{1}{2}n \right\rfloor$ or $\left\lfloor \frac{1}{2}(n-1) \right\rfloor$ simultaneous plane rotations. The convergence rate of [1JAC], however, does not necessarily match that of Kogbetliantz's algorithm.

Let

$$R_k = D_k + E_k + E_k^{\mathrm{T}},$$

and

$$S_k = R_k^{\mathrm{T}} R_k = \tilde{D}_k + \tilde{E}_k + \tilde{E}_k^{\mathrm{T}}, \tag{4.27}$$

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 (4.28) 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 Refs. [34, 35]). Thus, after a particular number of critical sweeps k_{cr}, we obtain

$$S_{k_{\mathrm{cr}}} = \begin{bmatrix} \boxed{T_1} & & & \\ & \boxed{T_2} & & \\ & & \boxed{T_3} & \\ & & & \ddots \\ & & & & \ddots \\ & & & & & \boxed{T_{n_{\mathrm{c}}}} \end{bmatrix} \tag{4.28}$$

so that the SVD of each T_i, $i = 1, 2, \ldots, n_{\mathrm{c}}$, 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 ith cluster. Since the quadratic convergence of Kogbetliantz's method for upper-triangular matrices [37] mirrors the quadratic convergence of the two-sided Jacobi method, [2JAC], for symmetric matrices having clustered spectra [38], we would obtain a faster global convergence for $k > k_{\mathrm{cr}}$ 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.

4.2.4 Block-Jacobi Algorithms

The above algorithms are well-suited for shared-memory computers. Although they can also be implemented on distributed memory systems, their efficiency on such systems may suffer due to communication costs. To increase the granularity of the computation (i.e., to increase the number of floating point operations between two communications), block algorithms are considered. For one-sided algorithms, each processor is allocated a block of columns instead of a single column. The computation remains the same as discussed above with the ordering of the rotations within a sweep is as given in Ref. [26].

For the two-sided version, the allocation manipulates two-dimensional blocks instead of single entries of the matrix. Some authors [39–41] propose a modification of the basic algorithm in which one annihilates, in each step, two off-diagonal blocks by performing a full SVD on a small-sized matrix. Good efficiencies are realized on distributed memory systems but this block strategy increases the number of sweeps needed to reach convergence.

We note that, parallel Jacobi algorithms can only surpass the speed of the bidiagonalization schemes of ScaLAPACK when the number of processors available are much larger than the size of the matrix under consideration.

4.3 METHODS FOR LARGE AND SPARSE MATRICES

4.3.1 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 Refs. [42, 43]. 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)` ($i = 1, \ldots, n$) is the index of the first nonzero entry of the ith 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, whereas 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 $A^{\mathrm{T}}A$ 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 Refs. [42, 44–51]. Current efficient packages for LU-factorization include UMFPACK [52], SuperLU [53], and MUMPS [54].

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 Refs. [43, 48, 55–58] or other references therein. When the matrix is symmetric positive definite, a preconditioned conjugate gradient scheme (PCG) may be an optimal choice as an iterative solver [2]. For symmetric indefinite systems, methods like SYMMLQ and MINRES [59] 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 because available iterative schemes cannot combine minimization of the residual, at any given step, for a given norm within the Krylov subspace, and orthogonalizing it with respect to the same subspace for some scalar product. Therefore, two classes of methods arise. The most popular methods include GMRES [60], Bi-CGSTAB [61], QMR [62], and TFQMR [63].

Before presenting methods for computing the sparse SVD, we note that classical methods for determining the SVD of dense matrices: the Golub–Kahan–Reinsch method [64, 65] and Jacobi-like SVD methods [34, 66] are not viable for large sparse matrices. Because 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 SVD.

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.

4.3.2 Subspace Iteration [SISVD]

Subspace iteration is perhaps one of the simplest algorithms used to solve large sparse eigenvalue problems. As discussed in Ref. [67], 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 [68] and if adapted to the matrix

$$\hat{B} = \gamma^2 I_n - A^{\mathrm{T}} A, \tag{4.29}$$

would involve forming the sequence

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

where γ^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. 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 Ref. [69]). 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 4.2. 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 θ_s/θ_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 Δ_k^2 in step (4) of Table 4.2.

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

$$\tilde{A}_{\text{aug}} = \begin{pmatrix} \gamma I & A \\ A^{\mathrm{T}} & \gamma I \end{pmatrix}, \tag{4.30}$$

where γ 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 $\tilde{A}_{\text{aug}}$ (see Section 4.1.1), 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 $\hat{B} = \gamma^2 I_n - A^{\mathrm{T}}A$ instead. In practice, γ can be chosen as either $\| A \|_1$ or $\| A \|_\infty$, depending upon the sparse data structure used to store the nonzero elements (row or columnwise). 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 $u_i = (1/\sigma_i)Av_i$ (see Section 4.1.1).

The orthogonal factorization in step (2) of Table 4.2 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 [70] or the block Householder orthogonalization method in step (2). To improve upon the two-sided Jacobi algorithm, originally suggested by Rutishauser [71] 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 Ref. [34]), is quite efficient for step (4) provided the dimension

TABLE 4.2
Subspace Iteration as Implemented in Rutishauser's ritzit [SISVD]

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

of the current subspace, s, is not too large. For larger subspaces, an optimized implementation of the classical EISPACK [72] pair, TRED2 and TQL2, or Cuppen's algorithm as parallelized by Dongarra and Sorensen [73] 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, $T_q(x/e)$, on the interval $[-e, e]$, where $e = \theta_s$ (assuming s vectors carried and $k = 1$ initially), $\xi_1 = 0.04$ and $\xi_2 = 4$:

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

where

$$\hat{q} = \begin{cases} 1, & \text{if } \theta_1 < \xi_1\theta_s \\ 2 \times \max\left[\dfrac{\xi_2}{\operatorname{arccosh}\left(\frac{\theta_s}{\theta_1}\right)}, 1\right] & \text{otherwise.} \end{cases} \tag{4.31}$$

The polynomial degree of the current iteration is then taken to be $q = q_{\text{new}}$. It can easily be shown that the strategy in (4.31) 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 ξ_1 and ξ_2 are treated as control parameters for determining the *frequency* and the *degree* of polynomial acceleration, respectively. In other words, large (small) values of ξ_1, inhibit (invoke) polynomial acceleration, and large (small) values of ξ_2 yield larger (smaller) polynomial degrees when acceleration is selected. Correspondingly, the number of matrix–vector multiplications will increase with ξ_2 and the total number of iterations may well increase with ξ_1. Controlling the parameters, ξ_1 and ξ_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 4.3.4.

4.3.3 Lanczos Methods

4.3.3.1 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. Let us consider the $(m + n) \times (m + n)$ 2-cyclic matrix A_{aug} given in (4.3), 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 [74]. Define $\beta_1 \equiv 0$ and $v_0 \equiv 0$, then for $i = 1, 2, \ldots, l$ define Lanczos vectors w_i and scalars α_i and β_{i+1} where

$$\begin{gathered} \beta_{i+1}w_{i+1} = A_{\text{aug}}w_i - \alpha_i w_i - \beta_i w_{i-1}, \text{ and } \alpha_i = w_i^{\mathrm{T}}(A_{\text{aug}}w_i - \beta_i w_{i-1}), \\ |\beta_{i+1}| = \|A_{\text{aug}}w_i - \alpha_i w_i - \beta_i w_{i-1}\|_2. \end{gathered} \tag{4.32}$$

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 β_{i+1} $(1 \leq i \leq (j-1))$, i.e.,

$$T_j \equiv \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \beta_j \\ & & & & \beta_j & \alpha_j \end{pmatrix}. \tag{4.33}$$

By definition, the vectors $\alpha_i w_i$ and $\beta_i w_{i-1}$ in (4.32) are respectively, the orthogonal projections of $A_{\text{aug}} w_i$ 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 $A_{\text{aug}} w_i$ with respect to w_i and w_{i-1}. The resulting α_i, β_{i+1} obtained in these orthogonalizations, define the corresponding Lanczos matrices. If we rewrite (4.32) in matrix form, then for each j we have

$$A_{\text{aug}} W_j = W_j T_j + \beta_{j+1} w_{j+1} e_j, \tag{4.34}$$

where $W_j \equiv [w_1, w_2, \ldots, w_j]$ is an $n \times j$ matrix whose kth column is the kth Lanczos vector, and e_j is the jth column of the $(m+n) \times (m+n)$ identity matrix. Thus, the Lanczos recursion (4.34) generates a family of real symmetric tridiagonal matrices related to both A_{aug} and w_1. Table 4.3 outlines the basic Lanczos procedure for computing the eigenvalues and eigenvectors of the symmetric 2-cyclic matrix A_{aug}.

As in subspace iteration, the matrix A_{aug} 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 [75], Parlett and Scott [76], and Simon [77] require additional Lanczos vectors to be readily accessible via secondary storage. We note that on a multiprocessor architecture, step (2) in Table 4.3 may benefit from any available optimized library routine that solves the symmetric tridiagonal eigenvalue problem (e.g., multisectioning in Ref. [78] or divide and conquer in Ref. [73]).

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 A_{aug}, and the appearance of spurious eigenvalues among the computed eigenvalues for some T_j. Approaches to

TABLE 4.3
Single-vector Lanczos Recursion [LASVD]

(1)	Use any variant of the Lanczos recursion (4.32) 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 A_{aug}, and hence singular values of A
(4)	For each eigenvalue λ 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 A_{aug} or the singular vectors of A

deal with these problems range between two different extremes. The first involves total reorthogonalization of every Lanczos vector with respect to every previously generated vector [79]. 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 [5] of the resulting spurious eigenvalues of A_{aug} (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 (4.32) equipped with a selective reorthogonalization strategy, `LANSO`, designed by Parlett and Scott [76] and Simon [77]. This particular method (`LASVD`) is primarily designed for the standard and generalized symmetric eigenvalue problem. We simply apply it to either $B = A^{\mathrm{T}}A$ or the 2-cyclic matrix A_{aug} defined in (4.3).

4.3.3.2 The Block Lanczos Method [BLSVD]

Here, we consider a block analog of the single-vector Lanczos method. Exploiting the structure of the matrix A_{aug} in (4.3), we can obtain an alternative form for the Lanczos recursion (4.32). If we apply the Lanczos recursion specified by (4.32) to A_{aug} with a starting vector $\tilde{u} = (u, 0)^{\mathrm{T}}$ 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 (4.32) reduces to the following: define $u_1 \equiv u$, $v_0 \equiv 0$, and $\beta_1 \equiv 0$, then for $i = 1, 2, \ldots, k$

$$\begin{aligned} \beta_{2i} v_i &= A^{\mathrm{T}} u_i - \beta_{2i-1} v_{i-1}, \\ \beta_{2i+1} u_{i+1} &= A v_i - \beta_{2i} u_i. \end{aligned} \tag{4.35}$$

The Lanczos recursion (4.35), 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 [80, 81], (4.35) can be represented in matrix form as

$$\begin{aligned} A^{\mathrm{T}} \hat{U}_k &= \hat{V}_k J_k^{\mathrm{T}} + Z_k, \\ A \hat{V}_k &= \hat{U}_k J_k + \hat{Z}_k, \end{aligned} \tag{4.36}$$

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

$$K_k \equiv \begin{pmatrix} O & J_k \\ J_k^{\mathrm{T}} & O \end{pmatrix}. \tag{4.37}$$

For the block analog of (4.36), 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 upper block bidiagonal matrix of order bk

$$J_k \equiv \begin{pmatrix} S_1 & R_1^{\mathrm{T}} & & & & \\ & S_2 & R_2^{\mathrm{T}} & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & R_{k-1}^{\mathrm{T}} \\ & & & & & S_k \end{pmatrix}, \tag{4.38}$$

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 ith singular triplet of J_k, then the approximation to the ith 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 C_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 C_k to diagonal form by a modified QR-algorithm. This diagonalization procedure is discussed in detail in Ref. [65]. The resulting diagonalized C_k will yield the approximate singular values of A, whereas 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, C_k. Golub et al. [82] 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 superdiagonal of J_k. In either reduction (bidiagonalization 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 (4.35) and (4.36) as our strategy for reducing the upper block bidiagonal matrix J_k to the bidiagonal form (C_k), i.e.,

$$\begin{aligned} J_k^{\mathrm{T}} \hat{Q} &= \hat{P} C_k^{\mathrm{T}}, \\ J_k \hat{P} &= \hat{Q} C_k, \end{aligned} \tag{4.39}$$

or

$$\begin{aligned} J_k p_j &= \alpha_j q_j + \beta_{j-1} q_{j-1}, \\ J_k^{\mathrm{T}} q_j &= \alpha_j p_j + \beta_j p_{j+1}, \end{aligned} \tag{4.40}$$

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 (4.40) require band matrix–vector multiplications which can be easily exploited by optimized level-2 BLAS routines [83] now resident in optimized mathematical libraries on most high-performance computers. For orthogonalization of the outermost Lanczos vectors, $\{U_i\}$ and

TABLE 4.4
Hybrid Lanczos Outer Iteration [BLSVD]

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

$\{V_i\}$, as well as the innermost Lanczos vectors, $\{p_i\}$ and $\{q_i\}$, we have chosen to apply a complete or total reorthogonalization [79] 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 Ref. [1].

As an alternative to the outer recursion defined by (4.36), which is derived from the equivalent 2-cyclic matrix A_{aug}, Table 4.4 depicts the simplified outer block Lanczos recursion for approximating the eigensystem of $A^{\mathrm{T}}A$. Combining the equations in (4.36), we obtain

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

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

$$H_k \equiv \begin{pmatrix} S_1 & R_1^{\mathrm{T}} & & & & \\ R_1 & S_2 & R_2^{\mathrm{T}} & & & \\ & R_2 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & R_{k-1}^{\mathrm{T}} \\ & & & & R_{k-1} & S_k \end{pmatrix}, \tag{4.41}$$

having block size b. We then apply the block Lanczos recursion [79] in Table 4.4 for computing the eigenpairs of the $n \times n$ symmetric positive definite matrix $A^{\mathrm{T}}A$. The tridiagonalization of H_k via an inner Lanczos recursion follows from simple modifications of (4.34). Analogous to the reduction of J_k in (4.38), 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 [69], which requires efficient dense matrix–vector multiplication (level-2 BLAS) routines [83] or efficient dense matrix–matrix multiplication (level-3 BLAS) routines [84], 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

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

4.3.4 The Trace Minimization Method [trsvd]

Another candidate subspace method for the SVD of sparse matrices is based upon the trace minimization algorithm discussed in Ref. [85] for the generalized eigenvalue problem

$$Hx = \lambda Gx, \tag{4.42}$$

where H and G are symmetric and G is also positive definite. To compute the SVD of an $m \times n$ matrix A, we initially replace H with $\tilde{A}_{\text{aug}}$ as defined in (4.30) or set $H = A^{\mathrm{T}}A$. Since we need to only consider equivalent standard symmetric eigenvalue problems, we simply define $G = I_{m+n}$ (or I_n if $H = A^{\mathrm{T}}A$). Without loss of generality, let us assume that $H = A^{\mathrm{T}}A$, $G = I_n$ 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 $Y^{\mathrm{T}}Y = I_p$, then using the Courant–Fischer theorem (see Ref. [86]) we obtain

$$\min_{Y \in \mathcal{Y}} \operatorname{trace}(Y^{\mathrm{T}}HY) = \sum_{i=1}^{p} \tilde{\sigma}_{n-i+1}, \tag{4.43}$$

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

$$Hz = \lambda z, \tag{4.44}$$

i.e.,

$$Y^{\mathrm{T}}HY = \tilde{\Sigma}, \quad Y^{\mathrm{T}}Y = I_p, \tag{4.45}$$

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

our trace minimization scheme [TRSVD] Ref. [85], see also [87], finds a sequence of iterates $Y_{k+1} = F(Y_k)$, where both Y_k and Y_{k+1} form a section of (4.44), and have the property trace $(Y_{k+1}^{\mathrm{T}} H Y_{k+1}) <$ trace $(Y_k^{\mathrm{T}} H Y_k)$. From (4.43), the matrix Y in (4.45) which minimizes trace$(Y^{\mathrm{T}}HY)$ is the matrix of H-eigenvectors associated with the p-smallest eigenvalues of the problem (4.44). As discussed in Ref. [85], $F(Y)$ can be chosen so that global convergence is assured. Moreover, (4.45) can be regarded as the quadratic minimization problem

$$\text{minimize } \operatorname{trace}(Y^{\mathrm{T}}HY) \tag{4.46}$$

subject to the constraints

$$Y^{\mathrm{T}}Y = I_p. \tag{4.47}$$

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

$$\begin{pmatrix} H & Y_k \\ Y_k^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L \end{pmatrix} = \begin{pmatrix} HY_k \\ 0 \end{pmatrix}, \tag{4.48}$$

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

$$\text{minimize } ((y_j^{(k)} - d_j^{(k)})^{\mathrm{T}} H (y_j^{(k)} - d_j^{(k)})) \tag{4.49}$$

subject to the constraints

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

where $d_j^{(k)} = \Delta_k e_j$, e_j the jth column of the identity, and $Y_k = [y_1^{(k)}, y_2^{(k)}, \ldots, y_p^{(k)}]$. The corrections Δ_k in this case are selected to be orthogonal to the previous estimates Y_k, i.e., so that (see Ref. [88])

$$\Delta_k^{\mathrm{T}} Y_k = 0.$$

We then recast (4.48) as

$$\begin{pmatrix} H & Y_k \\ Y_k^{\mathrm{T}} & 0 \end{pmatrix}\begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix}=\begin{pmatrix} Hy_j^{(k)} \\ 0 \end{pmatrix}, \quad j=1,2,\ldots,p, \tag{4.50}$$

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

The solution of the p-systems of linear equations in (4.50) 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 (4.50). As discussed in Refs. [1, 85], a major reason for using the conjugate gradient (CG) method for the solution of (4.50) 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 Δ_k.

4.3.4.1 Polynomial Acceleration Techniques for [TRSVD]

The Chebyshev acceleration strategy used within subspace iteration (see Section 4.3.2), 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

$$x=\frac{1}{P_q(\lambda)}P_q(H)x, \tag{4.51}$$

where $P_q(x)=T_q(x)+\epsilon I_n$, $T_q(x)$ is the Chebyshev polynomial of degree q and ϵ is chosen so that $P_q(H)$ is (symmetric) positive definite. The appropriate quadratic minimization problem similar to (4.49) here can be expressed as

$$\text{minimize } ((y_j^{(k)}-d_j^{(k)})^{\mathrm{T}}(y_j^{(k)}-d_j^{(k)})) \tag{4.52}$$

subject to the constraints

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

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 (4.50) becomes much easier to solve, i.e.,

$$\begin{pmatrix} I & P_q(H)Y_k \\ Y_k^{\mathrm{T}}P_q(H) & 0 \end{pmatrix}\begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix}=\begin{pmatrix} y_j^{(k)} \\ 0 \end{pmatrix}, \quad j=1,2,\ldots,p. \tag{4.53}$$

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

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

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

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

we have

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

where the polynomial degree, q, is determined by the strategy defined in Section 4.3.2.

4.3.4.2 Shifting Strategy for [TRSVD]

As discussed in Ref. [85], we can also accelerate the convergence of the Y_k's to eigenvectors of H by incorporating Ritz shifts (see Ref. [67]) into TRSVD. Specifically, we modify the symmetric eigenvalue problem in (4.44) as follows,

$$(H - \nu_j^{(k)} I)z_j = (\lambda_j - \nu_j^{(k)})z_j, \quad j = 1, 2, \ldots, s, \tag{4.54}$$

where $\nu_j^{(k)} = \tilde{\sigma}_{n-j+1}^{(k)}$ is the jth approximate eigenvalue at the kth iteration of [TRSVD], with λ_j, z_j an exact eigenpair of H. In other words, we simply use our most recent approximations to the eigenvalues of H from our kth section within [TRSVD] as Ritz shifts. As was shown by Wilkinson [89], the Rayleigh quotient iteration associated with (4.54) will ultimately achieve cubic convergence to the square of an exact singular value A, σ_{n-j+1}^2, provided $\nu_j^{(k)}$ is sufficiently close to σ_{n-j+1}^2. However, 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 Ref. [85] has been quite successful in maintaining global convergence with shifting.

Table 4.5 outlines the basic steps of [TRSVD]. The scheme appropriately utilizes polynomial (Chebyshev) acceleration before 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 (4.48) via (4.50) and the CG-algorithm. The *context switch* from either nonaccelerated (or polynomial-accelerated trace minimization iterations) to trace minimization iterations with Ritz shifting, is accomplished by monitoring the reduction of the residuals in Step (4) for isolated eigenvalues ($r_j^{(k)}$) or clusters of eigenvalues ($R_j^{(k)}$).

The Chebyshev acceleration and Ritz shifting within [TRSVD] for approximating smallest singular value of a 374×82 matrix with only 1343 nonzero elements has cut down the number

TABLE 4.5
[TRSVD] Algorithm with Chebyshev Acceleration and Ritz Shifts

(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 (4.45), or determine Y_k such that $Y_k^{\mathrm{T}} P_q(H) Y_k = I_p$, $Y_k^{\mathrm{T}} Y_k = \tilde{\Sigma}$
(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 (see Ref. [85]): For isolated eigenvalues: if $\|r_j^{(k)}\|_2 \le \eta \|r_j^{(k_0)}\|_2$, where $\eta \in [10^{-3}, 10^0]$ and $k_0 < k$ for some j For a cluster of eigenvalues (size c): if $\|R_j^{(k)}\|_F \le \eta \|R_j^{(k_0)}\|_F$, where $R_j^{(k)} \equiv \{r_j^{(k)}, \ldots, r_{j+c}^{(k)}\}$ and $k_0 < k$ for some j (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 (4.31) for $P_q(H)$ in iteration $k + 1$ (if needed)
(Step 7)	Update subspace iterate $Y_{k+1} \equiv Y_k - \Delta_k$ via (4.48) or (4.50)
(Step 8)	Set $k = k + 1$ and go to (Step 1)

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 three times higher than [TRSVD] with no acceleration.

4.3.5 Refinement of Left Singular Vectors

As discussed in Section 4.1.1, the smallest singular values of the matrix A lie in the interior of the spectrum of either the 2-cyclic matrix A_{aug} (4.3) or the shifted matrix $\tilde{A}_{\text{aug}}$ (4.30), all four candidate methods will have difficulties in approximating these singular values. The Lanczos-based methods, [LASVD] and [BLSVD] cannot be expected to effectively approximate the interior eigenvalues of either A_{aug} or $\tilde{A}_{\text{aug}}$. 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 (4.50), could indeed obtain accurate approximation of the smallest singular triplets. Recently, some progress has been realized by considering an implicitly restarted Lanczos bidiagonalization [90] that avoids direct inversion.

To target the p-smallest singular triplets of A, we compute the p-smallest eigenvalues and eigenvectors of the (operator) matrices listed in Table 4.6 for LASVD, BLSVD, and TRSVD. With SISVD, we determine the p-largest eigenpairs of $\gamma^2 I_n - A^{\mathrm{T}}A$. As discussed in Section 4.1.1, we may determine the singular values, σ_i, of A and their corresponding right singular vectors, v_i, as eigenpairs of either $A^{\mathrm{T}}A$ or $\gamma^2 I_n - A^{\mathrm{T}}A$, where A is $m \times n$ and $\gamma = \|A\|_{1,\infty}$. The corresponding left singular vector, u_i, must then be determined by

$$u_i = \frac{1}{\sigma_i} A v_i, \tag{4.55}$$

which may not be of sufficient accuracy (see Section 4.1.1) for a given precision in the residual (4.4). We point out, however, that alternative operators of the form (*shift and invert*)

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

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 Refs. [67, 69]). The effectiveness of this approach depends on an *accurate* approximation of the Cholesky factorization of $A^{\mathrm{T}}A$, or preferably of the QR-factorization of A, when A is large and sparse rectangular matrix.

TABLE 4.6
Operators of Equivalent Symmetric Eigenvalue Problems for Computing the *p*-Smallest Singular Triplets of the *m*×*n* Matrix *A*

Method	Label	Operator
Single-vector Lanczos	LASVD	$A^{\mathrm{T}}A$
Block Lanczos	BLSVD	$A^{\mathrm{T}}A$
Subspace iteration	SISVD	$\gamma^2 I_n - A^{\mathrm{T}}A$
Trace minimization	TRSVD	$A^{\mathrm{T}}A$

For [LASVD], we simply define $\tilde{A}_{\text{aug}} = A^{\text{T}}A$ in (4.32) and apply the strategy outlined in Table 4.3. For [SISVD] we make the substitution $\tilde{A}_{\text{aug}} = \gamma^2 I_n - A^{\text{T}}A$ for (4.30) and apply the method in Table 4.2. Similarly, we solve the generalized eigenvalue problem in (4.44) with

$$H = A^{\text{T}}A, \quad G = I_n,$$

and apply the strategy in Table 4.5 for the appropriate [TRSVD] method. Here, we do not require the shift, γ^2, because the trace minimization will converge to the smallest eigenvalues of $A^{\text{T}}A$ (and hence squares of singular values of A) by default. For [BLSVD], we combine the equations in (4.36) and obtain

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

where $H_k = J_k^{\text{T}} J_k$ is the $k \times k$ symmetric block tridiagonal matrix in (4.41), with the $n \times k$ matrix $\hat{Z}_k$ containing the remainder terms. The appropriate block Lanczos (outer) recursion is then given by Table 4.4.

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

$$\hat{r}_i = A^{\text{T}}A\tilde{v}_i - \tilde{\sigma}_i^2 \tilde{v}_i, \tag{4.56}$$

we must then obtain an approximation to the corresponding left singular vector, u_i, via (4.55). As mentioned in Section 4.1.1, it is quite possible that square roots of the approximate eigenvalues of either $A^{\text{T}}A$ or $\gamma^2 I_n - A^{\text{T}}A$ 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 (4.4), will be bounded by

$$\|r_i\|_2 \leq \|\hat{r}_i\|_2 / [\tilde{\sigma}_i (\|\tilde{u}_i\|_2^2 + \|\tilde{v}_i\|_2^2)^{\frac{1}{2}}], \tag{4.57}$$

where $\hat{r}_i$ is the residual given in (4.56) for the symmetric eigenvalue problem for $A^{\text{T}}A$ or $(\gamma^2 I_n - A^{\text{T}}A)$. 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, μ.

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

$$AA^{\text{T}}\tilde{u}_{i+1}^{(k)} = \tilde{\sigma}_i^2 \tilde{u}_i^{(k)}, \tag{4.58}$$

where $\tilde{u}_i^{(0)} \equiv u_i$ from (4.55). Since AA^{T} is symmetric semidefinite, direct methods developed by Aasen [91], Bunch and Kaufman [92], or Parlett and Reid [93] could be used in each step of (4.58) if AA^{T} is explicitly formed. With regard to iterative methods, the SYMMLQ algorithm developed by Paige and Saunders [94] 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.

$$\begin{pmatrix} \gamma I_n & A^{\text{T}} \\ A & \gamma I_m \end{pmatrix} \begin{pmatrix} v_i \\ u_i \end{pmatrix} = (\gamma + \sigma_i) \begin{pmatrix} v_i \\ u_i \end{pmatrix}, \tag{4.59}$$

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

$$\gamma = \min[1, \max\{\sigma_i\}]. \tag{4.60}$$

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

$$\begin{pmatrix} \gamma I_n & A^{\mathrm{T}} \\ A & \gamma I_m \end{pmatrix} \begin{pmatrix} \tilde{v}_i^{(k+1)} \\ \tilde{u}_i^{(k+1)} \end{pmatrix} = (\gamma + \tilde{\sigma}_i) \begin{pmatrix} \tilde{v}_i^{(k)} \\ \tilde{u}_i^{(k)} \end{pmatrix}, \tag{4.61}$$

By applying block Gaussian elimination to (4.61) we obtain a more optimal form (reduced system) of the recursion

$$\begin{pmatrix} \gamma I_n & A^{\mathrm{T}} \\ 0 & \gamma I_m - \frac{1}{\gamma} A A^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \tilde{v}_i^{(k+1)} \\ \tilde{u}_i^{(k+1)} \end{pmatrix} = (\gamma + \tilde{\sigma}_i) \begin{pmatrix} \tilde{v}_i^{(k)} \\ \tilde{u}_i^{(k)} - \frac{1}{\gamma} A \tilde{v}_i^{(k)} \end{pmatrix}. \tag{4.62}$$

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 (4.62), i.e.,

$$\left(\gamma I_m - \frac{1}{\gamma} A A^{\mathrm{T}}\right) \tilde{u}_i^{(k+1)} = (\gamma + \tilde{\sigma}_i) \left(\tilde{u}_i^{(k)} - \frac{1}{\gamma} A \tilde{v}_i\right), \tag{4.63}$$

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 (4.59), then (4.63) can be rewritten as

$$\left(\gamma I_m - \frac{1}{\gamma} A A^{\mathrm{T}}\right) \tilde{u}_i^{(k+1)} = (\gamma - \tilde{\sigma}_i^2/\gamma) \tilde{u}_i^{(k)}, \tag{4.64}$$

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 (4.63) is symmetric positive definite provided (4.60) holds. Accordingly, we may use parallel conjugate gradient iterations to refine each singular triplet approximation. Hence, the refinement procedure outlined in Table 4.6 may be considered as a *black box* procedure to follow the eigensolution of $A^{\mathrm{T}}A$ or $\gamma^2 I_n - A^{\mathrm{T}}A$ by any one of our four candidate methods for the sparse SVD. The iterations in Step (3) of the refinement scheme in Table 4.7 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.

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

TABLE 4.7
Refinement Procedure for the Left Singular Vector Approximations Obtained Via Scaling

(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 $\tilde{u}_i^{(0)} \equiv \frac{1}{\tilde{\sigma}_i} A \tilde{v}_i, \quad i = 1, 2, \ldots, p$
(3)	For $k = 0, 1, 2, \ldots$ (until $\|r_i\|_2 \leq$ tolerance or $k \geq k_{\max}$) Solve $\left(\gamma I_m - \frac{1}{\gamma} A A^{\mathrm{T}}\right) \tilde{u}_i^{(k+1)} = (\gamma - \tilde{\sigma}_i^2/\gamma) \tilde{u}_i^{(k)}$, $i = 1, 2, \ldots, p$ Set $\tilde{u}_i^{(k+1)} = \tilde{u}_i^{(k+1)} / \|\tilde{u}_i^{(k+1)}\|_2$

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 variety 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 reorthogonalization 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 $A^{\mathrm{T}}A$.

4.3.6 THE DAVIDSON METHODS

4.3.6.1 General Framework of the Methods

In 1975, Davidson [95] introduced a method for computing the smallest eigenvalues of the Schrödinger operator. Later, the method was generalized in two papers by Morgan and Scott [96] and Crouzeix et al. [97] in which the convergence of the method is proved. More recently, another version of the algorithm, under the name Jacobi–Davidson, was introduced in Ref. [98]. 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 Ref. [98] is strongly related to the trace minimization scheme discussed in Section 4.3.4, see Refs. [85, 99]. 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 $V_k = [v_1, ..., v_k]$ 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 $H_k = {V_k}^{\mathrm{T}} A V_k \in \mathbb{R}^{k\times k}$. 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 (λ, y) is an eigenpair of H_k, then it is expected that the Ritz pair (λ, 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 Refs. [85, 99]). 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 $k_{\max}$ 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 $k = 1 : k_{\max}$,
    compute $W_k = AV_k$;
    compute the interaction matrix $H_k = {V_k}^{\mathrm{T}} AV_k$;
    compute the smallest eigenpair (λ_k, y_k) of H_k ;
    compute the Ritz vector $x_k = V_k y_k$;
    compute the residual $r_k = W_k y_k - \lambda_k x_k$;
    if convergence then exit ;
    compute the new direction t_k **;** (Davidson correction)
    $V_{k+1} = \mathrm{MGS}(V_k, t_k)$;
  end ;
  until convergence ;
  $V_1 = \mathrm{MGS}(x_{k,1}, t_k)$;
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 [85] and [99].

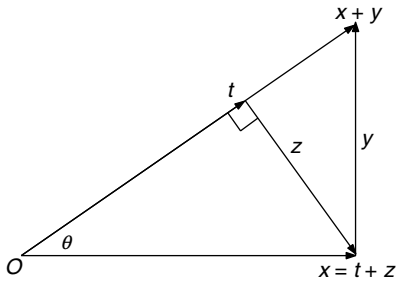
ALGORITHM : Generic Block Davidson
Choose an initial orthonormal matrix $V_1 = [v_1, \cdots, v_s] \in \mathbb{R}^{n \times s}$
Repeat
  for $k = 1 : k\max/s$,
    compute $W_k = AV_k$;
    compute the interaction matrix $H_k = {V_k}^{\mathrm{T}} AV_k$;
    compute the s smallest eigenpairs $(\lambda_{k,i}, y_{k,i})_{1 \le i \le s}$ 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 $(t_{k,i})_{1 \le i \le s}$ **;** (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 ;

4.3.6.2 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 $\rho(x)=x^{\mathrm{T}}Ax$ denotes the Rayleigh quotient of x) approximates the eigenvalue $\lambda+\delta$ which corresponds to the eigenvector $x+y$: $\lambda+\delta=\rho(x+y)=(x+y)^{\mathrm{T}}A(x+y)/\|x+y\|^2$. The quality of the approximation is measured by the norm of the residual $r=Ax-\lambda x$. Since $r=(I-xx^{\mathrm{T}})Ax$, the residual is orthogonal to vector x. Let us denote θ the angle $\angle(x,x+y)$; let t be the orthogonal projection of x onto $x+y$ and $z=t-x$.

Lemma 4.1 *The norms of the involved vectors are:*

$$\|x+y\| = \frac{1}{\cos^2\theta}, \tag{4.65}$$
$$\|y\| = \tan\theta, \tag{4.66}$$
$$\|z\| = \sin\theta, \tag{4.67}$$
$$\|t\| = \cos\theta. \tag{4.68}$$



Proof. Obvious. □

Proposition 4.5 *With the previous notations, the correction δ to the approximation λ of an eigenvalue, and the orthogonal correction y of the corresponding approximate eigenvector x, satisfy:*

$$\begin{cases} (A-\lambda I)y=-r+\delta(x+y), \\ y\perp x, \end{cases} \tag{4.69}$$

in which the following bounds hold:

$$|\delta| \leq 2\|A\|\tan^2\theta, \tag{4.70}$$
$$\|r\| \leq 2\|A\|\tan\theta\left(\frac{\sin\theta}{\cos^2\theta}+1\right). \tag{4.71}$$

Proof. Equation (4.69) is directly obtained from

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

Moreover

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

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

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

which proves (4.70). Bound (4.71) is a straight consequence of relations (4.69), (4.66), and (4.70). □

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

The system (4.69) is not easy to solve. Since the norm of the nonlinear term $\delta(x + y)$ is $O(\theta^2)$ whereas $\|r\| = O(\theta)$, one may instead consider the approximate problem

$$\begin{cases} (A - \lambda I)y = -r, \\ y \perp x, \end{cases} \tag{4.72}$$

which has no solution except the trivial solution when λ 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:

$$(A - \lambda I)y = -r, \tag{4.73}$$

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 (4.72) onto the hyperplane $(x)^{\perp}$ orthogonal to x.

More precisely, the different approaches for the correction are:

Davidson with a preconditionner: solve

$$(M - \lambda I)y = -r, \tag{4.74}$$

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 λ is close to an eigenvalue, solve approximately

$$(A - \lambda I)y = -r, \tag{4.75}$$

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:

$$\begin{cases} Q(x)(A - \lambda I)y = -r, \\ y \perp x, \end{cases} \tag{4.76}$$

where $Q(x) = I - xx^{\mathrm{T}}$ 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 Eldén [100] 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. Simoncini and Eldén [100] also provides a stopping criterion for controlling the inner iterations of an iterative solver for the correction vectors.

4.3.6.3 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 $C = A^{\mathrm{T}}A$, or to obtain the innermost positive eigenvalue of the 2-cyclic augmented matrix in (4.3). 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 $H_k = {V_k}^{\mathrm{T}} C V_k = A{V_k}^{\mathrm{T}} A V_k$, we still have to compute the residuals corresponding to the Ritz pairs which do not 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

$$A^{\mathrm{T}} A t_k \quad = \quad r_k. \tag{4.77}$$

Obtaining an exact solution of (4.77) 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 (4.77) without any shifts; the benefit is that we deal with a fixed symmetric positive definite system matrix.

The approximate solution of (4.77) 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 Ref. [101] in which M is obtained from approximate factorizations:

Incomplete LU-factorization of A: Here, $M = U^{-1}L^{-1}L^{-\mathrm{T}}U^{-\mathrm{T}}$, 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, $M = R^{-1}R^{-\mathrm{T}}$, 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 $A^{\mathrm{T}}A$: Here, $M = L^{-\mathrm{T}}L^{-1}$, 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 Ref. [101] 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 $A^{\mathrm{T}}A$ to compute the smallest eigenvalue and the corresponding eigenvector. More recently, the Jacobi–Davidson method has been adapted in Ref. [102] for obtaining the singular values of A by considering the eigenvalue problem 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 whereas 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).

4.4 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.

4.4.1 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 Ref. [103] which considers a matrix stored using the CSR format outlined in Section 4.3.1.

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:

$$A = \begin{bmatrix} A^0 \\ \hline A^1 \\ \hline \vdots \\ \hline A^{p-1} \end{bmatrix} \quad v = \begin{bmatrix} v_0 \\ \hline v_1 \\ \hline \vdots \\ \hline v_{p-1} \end{bmatrix} \begin{array}{l} \longrightarrow \text{Node } 0 \\ \longrightarrow \text{Node } 1 \\ \longrightarrow \vdots \\ \longrightarrow \text{Node } (p-1) \end{array}$$

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 $A_{\text{loc},k}$ defines the component which do not require communication during the multiplication. On the same processor, the exterior part is split into the blocks $A^j_{\text{ext},k}$ $(j \neq k)$ as shown below.

Data partition for the multiplications by A_z and A_z^H:

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

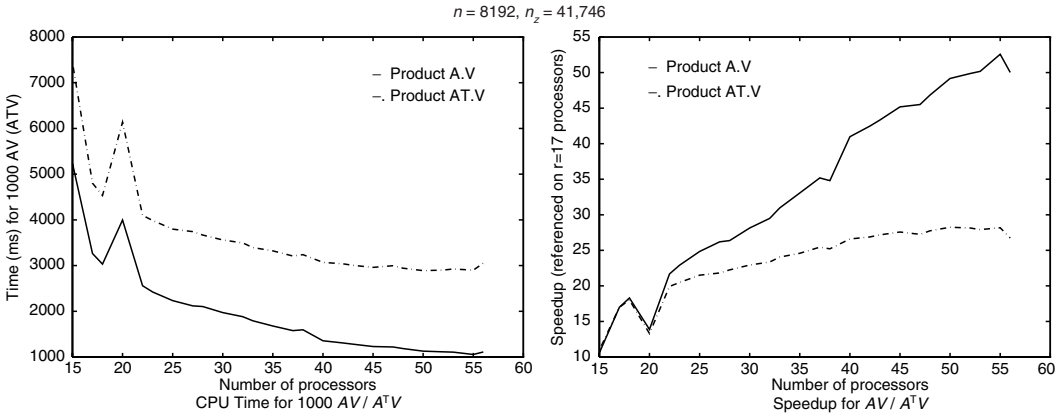


FIGURE 4.2 Timings for matrix–vector multiplications.

Algorithms for the matrix–vector multiplication

The algorithms for the two matrix–vector multiplications are expressed in the two following tables in which the communicating routine Send is preferably a nonblocking one.

k = processor number; p = number of processors.

Matrix–vector multiplication $v \rightarrow Av$**:**

Step 1: Send the needed components of v^k_{loc} to other processors.
Step 2: Compute $w^k = A_{\text{loc},k} v^k_{\text{loc}}$.
Step 3: Receive the needed components of v^j_{loc} for $j \neq k$ from other processors.
Step 4: Compute $w^k = w^k + \sum_{j \neq k} A^j_{\text{ext},k} v^j_{\text{loc}}$.

Matrix transpose–vector multiplication $v \rightarrow A^{\text{T}}v$ **:**

Step 1: Compute $y^j = A^{j\text{T}}_{\text{ext}} v^k_{\text{loc},j}$ for $j \neq k$.
Step 2: Send y^j for $j \neq k$ to node number j.
Step 3: Compute $w^k = A^{\text{T}}_{loc,k} v^k_{\text{loc}}$.
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 [14] test suite was used to measure the performance of the above two matrix–vector multiplication schemes. Run times and speedups are displayed in Table 4.8, showing that one should avoid matrix-vector multiplications involving the transpose of the stored matrix, at least on architectures similar to the Intel Paragon.

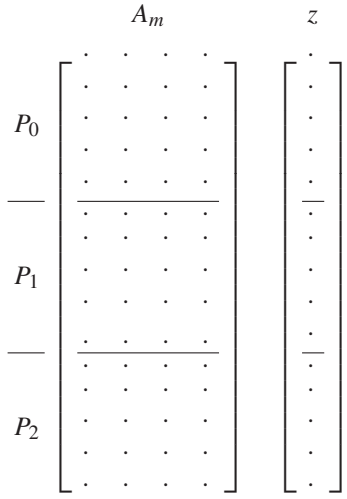


TABLE 4.8 Distribution by contiguous rows

4.4.2 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 MGS-algorithm, does not offer effective parallelism for the case at hand $(m \gg n)$. The classical version is often considered due to its suitability on parallel architectures. In this case, however, to maintain numerical reliability, it is necessary to reapply the procedure a second time. We present here a better procedure that we outlined in a previous study, e.g., see Ref. [104], 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 the vector Qy for any vector y. Assuming that the columns of A are distributed on a ring of r processors, see Table 4.8, the two procedures are given below.

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 through 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 speedup when $(n/r) \gg m$ and $m \gg r$.

Experiment. The following experiment illustrates the superior scalability of the scheme ROC (ROC=ROCDEC+ROCVEC) with respect to the MGS procedure. The goal is to compute Qy where Q is the Q-factor in the QR-factorization of A. In Table 4.9, 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.

TABLE 4.9
Scalability of the Orthogonalization Process (As Illustrated by the Running Time in Seconds)

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

4.4.3 Computing the Smallest Singular Value on Several Processors

As mentioned earlier (see Proposition 4.1) the smallest singular value of A can be computed from the largest eigenvalue of the matrix

$$B = \begin{pmatrix} 0 & R^{-1} \\ R^{-T} & 0 \end{pmatrix}, \tag{4.78}$$

where $A = \mathrm{QR}$ 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 multifrontal QR-decomposition (MFQRD), presented in Ref. [105], 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 (rowwise) adequate for parallel solvers. For a detailed description of this procedure, the reader is refered to Ref. [13].

Table 4.10 shows the wall-clock time needed to compute $\sigma_{\min}$ for three test matrices $S_1 \in \mathbb{R}^{1890\times1890}$, $S_2 \in \mathbb{R}^{3906\times3906}$, and $S_3 \in \mathbb{R}^{32130\times32130}$. 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

TABLE 4.10
Parallel Computation of $\sigma_{\min}$ on a Small Cluster of Machines.

Matrix	1 Proc.	3 Procs.	Speedup	Efficiency
Multi-frontal QR-decomposition				
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				
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
$\sigma_{\min}$				
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

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 of 61%.

4.5 APPLICATION: PARALLEL COMPUTATION OF A PSEUDOSPECTRUM

As discussed in Section 4.1.3, the computation of the pseudospectrum 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

$$\Gamma_\epsilon = \{z \in \mathbb{C} \mid \sigma_{\min}(A - zI) = \epsilon\}, \tag{4.79}$$

are of much less complexity than methods based on grid discretization [106]. The first attempt in this direction was published by Brühl [11]. Based on a continuation with a predictor–corrector scheme, the process may fail for angular discontinuities along the level curve [12, 107]. Trefethen and Wright [108] use the upper Hessenberg matrix constructed after successive iterations of the implicitly restarted Arnoldi algorithm to cheaply compute an approximation of the pseudospectrum. However, they show that for highly nonnormal matrices the computed pseudospectrum is a low approximation of the exact one.

4.5.1 PARALLEL PATH-FOLLOWING ALGORITHM USING TRIANGLES

In this section, we present the parallel path-following algorithm using triangles (PPAT) for computing the level curve Γ_ϵ. For a detailed description of this algorithm, the reader is referred to Refs. [13, 109].

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.

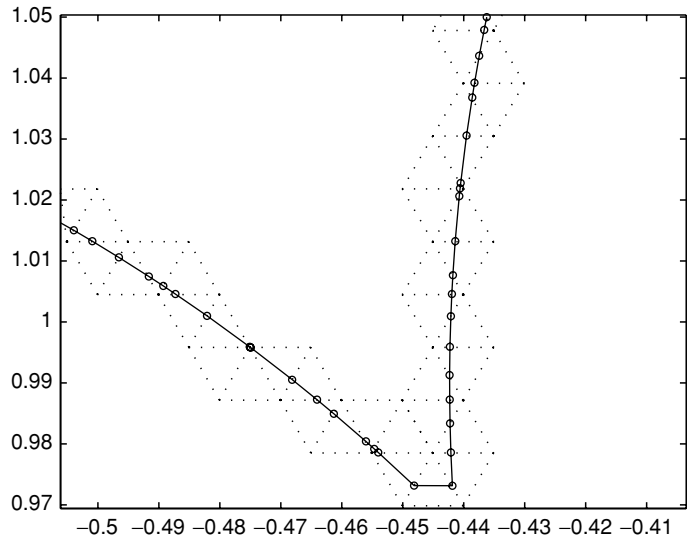


FIGURE 4.3 Computing a level curve using PPAT.

The main idea is to line up a set of equilateral triangles along the level curve as presented in Figure 4.3 and use a bisection algorithm to compute a numerical approximation of the pseudo-spectrum. More specifically, 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 $T_1, \ldots, T_n$ such that $T_n = T_0$ is defined. In Ref. [109], it is proven that to compute a level curve of length l, the number of equilateral triangles of size τ satisfies

$$\frac{l}{\tau} \leq n \leq \frac{10l}{\sqrt{3}\tau}. \tag{4.80}$$

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} whereas 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.

4.5.2 Speedup and Efficiency

It is shown in Ref. [13] 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

$$S_p = \begin{cases} \dfrac{2n(q+1)}{n+2q} & \text{if } p \geq 2q+2, \\ \dfrac{2np(q+1)}{p^2-2p+2n(q+1)} & \text{if } p < 2q+2. \end{cases} \tag{4.81}$$

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 $p \geq 2q+2$.

4.5.3 Test Problems

Three test matrices were selected from the Matrix Market suite [14]. Table 4.11 summarizes their characteristics (Nz is the number of nonzero 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, 600 MHz, 128 MB RAM).

Figure 4.4 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 s, the level curve $\epsilon = 0.05$ of `Olm1000` split into 15 slices. A single processor required 4020 s 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 because 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.

TABLE 4.11
Test Matrices from the Non-Hermitian Eigenvalue problem (NEP) Collection

Matrix	Order	Nz	$\|\|A\|\|_F$	$t_{\sigma_{\min}}$ (s)
Olm1000	1,000	3,996	1.3×10^6	0.27
Rdb3200L	3,200	18,880	2.8×10^3	15.14
Dw8192	8,192	41,746	1.6×10^3	114.36

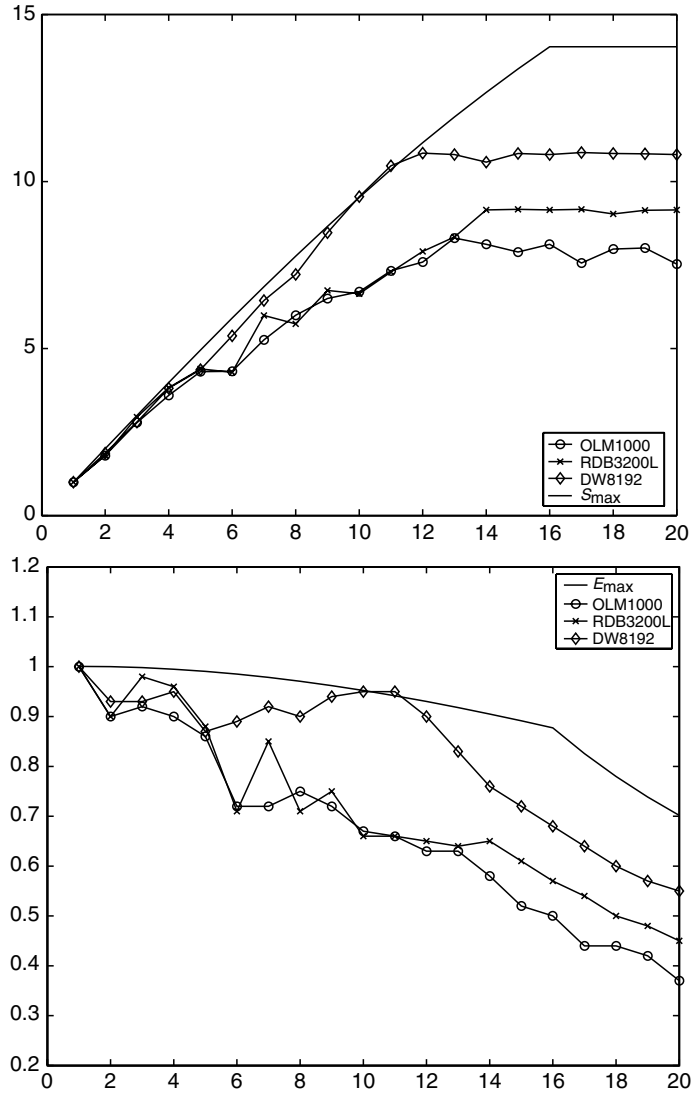


FIGURE 4.4 Speedup and efficiency of PPAT.

REFERENCES

[1] M.W. Berry. Large scale singular value computations. *Int. J. Supercomputer Appl.*, 6:13–49, 1992.

[2] G.H. Golub and C.F.Van Loan. *Matrix Computations, 3rd* ed. John Hopkins University Press, Baltimare, MD, 1996.

[3] G.W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990.

[4] J.-G. Sun. Condition number and backward error for the generalized singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 22(2):323–341, 2000.

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

[6] J. Demmel, M. Gu, S. Eisenstat, Z. Drmač, I. Slapničar, and K. Veselić. Computing the singular value decomposition with high accuracy. *Linear Algebra Appl.*, 299(1–3):21–80, 1999.
[7] J.-G. Sun. A note on simple non-zero singular values. *J. Comput. Math.*, (6):258–266, 1988. AQ1
[8] 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. Geosci.*, 6:161–194, 2002.
[9] L.N. Trefethen. Pseudospectra of matrices. In D.F. Griffiths and G.A. Watson, Eds., *Numerical Analysis*, Dundee 1991. Longman, Chicago, 1992, pp. 234–266.
[10] S.K. Godunov. Spectral portraits of matrices and criteria of spectrum dichotomy. In L. Atanassova and J. Hezberger, Eds., *3rd Int. IMACS-CAMM Symp. Comp. Arithmetic Enclosure Meth.*, Amsterdam, 1992.
[11] M. Brühl. A curve tracing algorithm for computing the pseudospectrum. *BIT*, 36(3): 441–454, 1996.
[12] C. Bekas and E. Gallopoulos. Cobra: parallel path following for computing the matrix pseudospectrum. *Parallel Comput.*, 27(14):1879–1896, 2001.
[13] D. Mezher and B. Philippe. Parallel computation of pseudospectra of large sparse matrices. *Parallel Comput.*, 28(2):199–221, 2002.
[14] Matrix Market. http://math.nist.gov/MatrixMarket/.
[15] L.S. Blackford, J. Choi, A. Cleary, E.D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.
[16] ScaLAPACK. The scalapack project. http://www.netlib.org/scalapack/.
[17] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, Y.-J.J. Wu, and R. van de Geijn. PLAPACK: parallel linear algebra package. In *P. SIAM Parallel Process. Conf.*, 1997.
[18] PLAPACK. PLAPACK: parallel linear algebra package.
[19] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.-M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to scilab//, the ouragan project. *Parallel Comput.*, 11(27):1497–1519, 2001.
[20] Mathtools.net. Mathtools.net > matlab > parallel. http://www.mathtools.net/MATLAB/Parallel/.
[21] G. Forsythe and P. Henrici. The cyclic Jacobi method for computing the principal values of a complex matrix. *Trans. Am. Math. Soc.*, 94:1–23, 1960.
[22] A. Schonhage. Zur konvergenz des Jacobi-Verfahrens. *Numer. Math.*, 3:374–380, 1961.
[23] J.H. Wilkinson. Note on the quadratic convergence of the cyclic jacobi process. *Numer. Math.*, 5: 296–300, 1962.
[24] A. Sameh. On Jacobi and Jacobi-like algorithms for a parallel computer. *Math. Comput.*, 25:579–590, 1971.
[25] F. Luk and H. Park. A proof of convergence for two parallel Jacobi SVD algorithms. *IEEE Trans. Comput.*, 38(6):806–811, 1989.
[26] F. Luk and H. Park. On parallel Jacobi orderings. *SIAM J. Sci. Stat. Comput.*, 10(1):18–26, 1989.
[27] P. Henrici. On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing eigenvalues of Hermitian matrices. *Soc. Ind. Appl. Math.*, 6:144–162, 1958.
[28] A. Sameh. *Solving the Linear Least Squares Problem on a Linear Array of Processors*. Academic Press, New York, 1985, pp. 191–200.
[29] 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.
[30] J.C. Nash. A one-sided transformation method for the singular value decomposition and algebraic eigenproblems. *Comput. J.*, 18(1):74–76, 1975.
[31] F.T. Luk. Computing the singular value decomposition on the Illiac IV. *ACM Trans. Math. Soft.*, 6(4):524–539, 1980.
[32] R.P. Brent and F.T. Luk. The solution of singular value and symmetric eigenproblems on multiprocessor arrays. *SIAM J. Sci. Stat.*, 6:69–84, 1985.
[33] R.P. Brent, F.T. Luk, and C. Van Loan. Computation of the singular value decomposition using mesh connected processors. *VLSI Comput. Syst.*, 1(3):242–270, 1985.

[34] M.W. Berry and A.H. Sameh. An overview of parallel algorithms for the singular value and symmetric eigenvalue problems. *Comput. Appl. Math.*, 27:191–213, 1989.
[35] 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.
[36] 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.
[37] J.-P. Charlier and P. Van Dooren. On Kogbetliantz's SVD algorithm in the presence of clusters. *Linear Algebra Appl.*, 95:135–160, 1987.
[38] J.H. Wilkinson. Almost diagonal matrices with multiple or close eigenvalues. *Linear Algebra Appl.*, 1:1–12, 1968.
[39] M. Bečka and M. Vajtešic. Block–Jacobi SVD algorithms for distributed memory systems I. *Parallel Algorithms Appl.*, 13:265–267, 1999.
[40] M. Bečka and M. Vajtešic. Block–Jacobi SVD algorithms for distributed memory systems II. *Parallel Algorithms Appli.*, 14:37–56, 1999.
[41] M. Bečka, G. Okša, and M. Vajtešic. Dynamic ordering for a Block–Jacobi SVD algorithm. *Parallel Comput.*, 28:243–262, 2002.
[42] I. Duff, A. Erisman, and J Reid. *Direct Methods for Sparse Matrices.* Oxford Science Publications, Oxford, 1992.
[43] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, MA 1996.
[44] P.R. Amestoy, I.S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numer. Linear Algebra Appl.*, 3:275–300, 1996.
[45] P.R. Amestoy, I.S. Duff, and J.Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Meth. Appl. Mech. Eng.*, 184(2–4):501–520, 2000.
[46] I. Brainman and S. Toledo. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 23(4):998–1012, 2002.
[47] T.A. Davis and I.S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Software*, 25:1–19, 1999.
[48] 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, Philadelphia, 1991.
[49] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Joseph. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999.
[50] I.S. Duff. A review of frontal methods for solving linear systems. *Comput. Phys. Commun.*, 97(1–2):45–52, 1996.
[51] 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.
[52] T.A. Davis. Umfpack version 4.0. http://www.cise.ufl.edu/research/sparse/umfpack, April 2002.
[53] X.S. Li. SuperLU version 2. http://www.nersc.gov/ xiaoye/SuperLU/, September 1999.
[54] 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.
[55] S.F. Ashby, T.A. Manteuffel, and P.E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 26:1542–1568, 1990.
[56] 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 ed*. SIAM / netlib, Philadelphia, PA, 1994.
[57] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, Vol. 95 of *Applied Mathematical Sciences*. Springer-Verlag, Heidelberg 1994.
[58] G. Meurant. *Computer Solution of Large Linear Systems*. North-Holland, Amsterdam, 1999.
[59] C. Paige and M. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J Numeri. Anal.*, 12:617–629, 1975.
[60] Y. Saad and H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.
[61] H.A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[62] R. Freund and N. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.
[63] R. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. comput.*, 14:470–482, 1993.
[64] 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.
[65] G.H. Golub and C. Reinsch. *Singular Value Decomposition and Least Squares Solutions*. Springer-Verlag, Heidelberg, 1971.
[66] M.R. Hestens. Inversion of matrices by biorthogonalization and related results. *Soc. Ind. Appl. Math.*, 6:51–90, 1958.
[67] B.N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, New York, 1980.
[68] F.L. Bauer. Das verfahred der treppeniteration und verwandte verfahren zur losung algebraischer eigenwertprobleme. *ZAMP*, 8:214–235, 1957.
[69] H. Rutishauser. Simultaneous iteration method for symmetric matrices. *Numer. Math.*, 16:205–223, 1970.
[70] 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.
[71] H. Rutishauser. Computational aspects of F.L. Bauer's simultaneous iteration method. *Numer. Math.*, 13:4–13, 1969.
[72] B. Smith, J. Boyce, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, and C. Moler. *Matrix Eigensystem Routines — EISPACK Guide*, 2nd ed. Springer-Verlag, Heidelberg, 1976.
[73] J. Dongarra and D. Sorensen. A fast algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8(2):s139–s154, 1987.
[74] C.C. Paige. Error analysis of the Lanczos algorithms for tridiagonalizing a symmetric matrix. *Inst. Math. Appl.*, 18:341–349, 1976.
[75] J. Grcar. Analysis of the Lanczos Algorithm and of the Approximation Problem in Richardson's Method, Ph.D. Thesis. Technical report, The University of Illinois, Chicago, 1981.
[76] B.N. Parlett and D.S. Scott. The Lanczos algorithm with selective reorthogonalization. *Math. Comput.*, 33:217–238, 1979.
[77] H. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra Appl.*, 61:101–131, 1984.
[78] 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.
[79] G.H. Golub and C. Van Loan. *Matrix Computations*, 2nd ed. Johns Hopkins University Press, Battimore, MD, 1989.
[80] R.R. Underwood. An Iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems, Ph.D. thesis. Technical report, Stanford University, Stanford, 1975.
[81] G.H. Golub and R.R. Underwood. *The Block Lanczos Method for Computing Eigenvalues*, Academic Press, New York, 1977, pp. 361–377.
[82] 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. Math. Software*, 7(2):149–169, 1981.
[83] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Software*, 14(1):1–17, 1988.
[84] 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.
[85] 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.
[86] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.
[87] M.W. Berry, B. Parlett, and A.H. Sameh. Computing extremal singular triplets of sparse matrices on a shared-memory multiprocessor. *Int. J. High Speed Comput.*, 2:239–275, 1994.
[88] D.G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Reading, Massachusetts, 1973.

[89] J.H. Wilkinson. *Inverse Iteration in Theory and in Practice*. Academic Press, New York, 1972.
[90] E. Kokiopoulou, C. Bekas, and E. Gallopoulos. Computing Smallest Singular Triplets with Implicitly Restarted Lanczos Bidiagonalization. Technical report, Computer Eng. & Informatics Dept., HPCLAB-TR-17-02-03. Appl. Numer. Math., 2003, to appear.
[91] J.O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.
[92] J.R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comput.*, 31:162–179, 1977.
[93] 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.
[94] C.C. Paige and M.A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.
[95] 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.
[96] R.B. Morgan and D.S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Stat. Comput.*, 7:817–825, 1986.
[97] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson method. *SIAM J. Sci. Stat. Comput.*, 15:1: 62–76, 1994.
[98] G.L.G. Sleipen and H.A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996. Also SIGEST in *SIAM Rev*. 42(2): 267–293.
[99] A. Sameh and Z. Tong. The trace minimization method for the symmetric generalized eigenvalue problem. *J. Comput. Appl. Math.*, 123:155–175, 2000.
[100] V. Simoncini and L. Eldén. Inexact Rayleigh quotient-type methods for eigenvalue computations. *BIT*, 42(1):159–182, 2002.
[101] B. Philippe and M. Sadkane. Computation of the fundamental singular subspace of a large matrix. *Linear Algebra Appl.*, 257:77–104, 1997.
[102] M.E. Hochstenbach. A Jacobi–Davidson type SVD method. *SIAM J. Sci. Comput.*, 23(2):606–628, 2001.
[103] 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.
[104] R. Sidje and B. Philippe. Parallel Krylov subspace basis computation. In J. Tankoano, Ed., *CARI'94 Proc., Ouagadougou*. INRIA — ORSTOM, ORSTOM Editions, Paris, 1994, pp. 421–440.
[105] P.R. Amestoy, I.S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numer. Linear Algebra Appl.*, 3:275–300, 1996.
[106] L.N. Trefethen. Computation of pseudospectra. *Acta Numer.*, 247–295, 1999.
[107] S.H. Lui. Computation of pseudospectra by continuation. *SIAM J. Sci. Comput.*, 28(2): 565–573, 1997.
[108] L.N. Trefethen and T. Wright. Large-scale Computation of Pseudospectra using ARPACK and Eigs. Technical report, Oxford University Computing Laboratory, Oxford, June 2000.
[109] D. Mezher and B. Philippe. PAT — a reliable path-following algorithm. *Numer. Algorithms*, 29: 131–152, 2002.

AUTHOR QUERIES

AQ1 Please provide vol. number
AQ2 Please update
AQ3 Please provide Vol. number