SVDPACKC (Version 1.0) USER'S GUIDE

Michael Berry, Theresa Do, Gavin O'Brien
Vijay Krishna, and Sowmini Varadhan

Computer Science Department

CS-93-194 April 1993

(Revised October 1993)

SVDPACKC (Version 1.0) User's Guide[1]

Michael Berry [2] Theresa Do[2] Gavin O'Brien[2]
Vijay Krishna[2] Sowmini Varadhan[2]

October 6, 1993

[1]This research was supported by the National Science Foundation under grant NSF CDA-9115428, and by Apple Computer Inc., Cupertino, CA, under contract C24-9100120. This document is also available as Department of Computer Science Technical Report No. CS-93-194, University of Tennessee, May 1993.

[2]Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN, 37996-1301.

Abstract

SVDPACKC comprises four numerical (iterative) methods for computing the singular value decomposition (SVD) of large sparse matrices using ANSI C. This software package implements Lanczos and subspace iteration-based methods for determining several of the largest singular triplets (singular values and corresponding left- and right-singular vectors) for large sparse matrices. The package has been ported to a variety of machines ranging from supercomputers to workstations: CRAY Y-MP, IBM RS/6000-550, DEC 5000-100, HP 9000-750, SPARCstation 2, and Macintosh II/fx. This document (*i*) explains each algorithm in some detail, (*ii*) explains the input parameters for each program, (*iii*) explains how to compile/execute each program, and (*iv*) illustrates the performance of each method when we compute lower rank approximations to sparse *term-document* matrices from information retrieval applications. A user-friendly software interface to the package for UNIX-based systems and the Macintosh II/fx is also described.

Contents

1 Introduction

The singular value decomposition (SVD) is commonly used in the solution of unconstrained linear least squares problems, matrix rank estimation, and canonical correlation analysis. In applications such as information retrieval, seismic reflection tomography, and real-time signal processing, the solution to these problems is needed in the shortest possible time. Given the growing availability of high performance computer systems, there has been great interest in the development of efficient implementations of the singular value decomposition, in general. In applications such as information retrieval ([12], [8]), the data matrix whose SVD is sought is usually large and sparse. It is this particular case that motivated the original Fortran-77 SVDPACK library [5]. SVDPACKC is a more portable ANSI C implementation of SVD methods which can be used to determine singular values and singular vectors of large sparse matrices on a variety of machines. SVDPACKC uses Lanczos, block-Lanczos, subspace iteration, and trace minimization methods for determining several of the largest singular values and corresponding singular vectors for *unstructured* sparse matrices arising from practical applications. Before discussing specific SVDPACKC routines, we make a few definitions, and review a few of the fundamental characterizations of the SVD.

Without loss of generality, suppose A is a sparse m by n ($m \gg n$) matrix with rank(A) = r. The singular value decomposition (SVD) of A can be defined as

$$A = U\Sigma V^T, \tag{1}$$

where $U^TU = V^TV = I_n$ and $\Sigma = diag(\sigma_1, ..., \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r$, $\sigma_i = 0$ for $i \geq r+1$. The first r columns of the orthogonal matrices U and V define the orthonormalized eigenvectors associated with the r nonzero eigenvalues of AA^T and A^TA, respectively. The singular values of A are defined as the diagonal elements of Σ which are the nonnegative square roots of the n eigenvalues of AA^T. The set $\{u_i, \sigma_i, v_i\}$ is called the i-th singular triplet. The singular vectors (triplets) corresponding to large (small) singular values are called large (small) singular vectors (triplets). The development of SVDPACKC was primarily motivated by the following problem:

Given the sparse $m \times n$ matrix A and $p < n$, determine the p-largest singular triplets of A as defined by (1).

To illustrate ways in which the SVD can reveal important information about the structure of a matrix we state two well-known theorems:

Theorem 1.1 *Let the SVD of A be given by (1) and*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0 ,$$

and let $R(A)$ and $N(A)$ denote the range and null space of A, respectively, then

1. *Rank property:* $\mathrm{rank}(A) = r$, $\mathrm{N}(A) \equiv span\{v_{r+1}, \cdots, v_n\}$, *and* $\mathrm{R}(A) \equiv span\{u_1, \cdots, u_r\}$, *where* $U = [u_1\ u_2\ \cdots\ u_m]$ *and* $V = [v_1\ v_2\ \cdots\ v_n]$.

2. *Dyadic decomposition:* $A = \sum_{i=1}^{r} u_i \cdot \sigma_i \cdot v_i^T$.

3. *Norms:* $\|A\|_F^2 = \sigma_1^2 + \cdots + \sigma_r^2$, *and* $\|A\|_2^2 = \sigma_1$.

The rank property, perhaps one of the most valuable aspects of the SVD, allows us to use the singular values of A as quantitative measures of the qualitative notion of rank. The dyadic decomposition, which is the rationale for data reduction or compression in many applications, provides a canonical description of a matrix as a sum of r rank-one matrices of decreasing importance, as measured by the singular values. The three results in Theorem 1.1 can be combined to yield the following quantification of matrix rank deficiency (see [18] for a proof):

Theorem 1.2 [Eckart and Young] *Let the SVD of A be given by (1) with $r = rank(A) \leq p = min(m,n)$ and define:*

$$A_k = \sum_{i=1}^{k} u_i \cdot \sigma_i \cdot v_i^T \ \textit{with}\ k < r ,$$

then

$$\min_{\mathrm{r}(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sigma_{k+1}^2 + \cdots + \sigma_p^2 .$$

This important result, which indicates that A_k is the best rank-k approximation (in a least squares sense) to the matrix A, is the basis for concepts such as *data reduction* and *image enhancement*. In fact, A_k is the best approximation to A for any unitarily invariant norm ([24]). Hence,

$$\min_{\mathrm{r}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}.$$

In the next section, we illustrate the applicability of Theorem 1.2 to problems in information retrieval which motivated the development of SVDPACKC. In Section 3, we present two pairs of Lanczos-based routines, (las1, las2) and (bls1, bls2), and two pairs of subspace iteration-based routines, (sis1, sis2) and (tms1, tms2), for solving equivalent sparse symmetric eigenvalue problems. A script-based user interface for SVDPACKC, which can be used for test and production runs, is described in Section 4, and in Section 5, we illustrate the performance of SVDPACKC on a few workstations using sparse matrices collected from Apple Computer Inc. in information retrieval applications. We conclude with a brief list of future enhancements to SVDPACKC in Section 6.

2 Applications

Sparse linear least squares problems naturally arise in many *real*-world applications. The use of the sparse SVD to solve such problems is of current interest to researchers in fields such as query-based information retrieval and seismic reflection tomography. In this section we will briefly focus on the use of SVDPACKC relative to information retrieval models. See [4] for a discussion of the role SVD plays in seismic tomography applications.

In [8] and [12] a new approach to automatic indexing and retrieval is discussed. It is designed to overcome a fundamental problem that plagues existing information retrieval techniques that try to match words of queries with words of documents. The problem is that users want to retrieve on the basis of conceptual topic or meaning of a document. There are usually many ways to express a given concept (*synonymy*), so the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings (*polysemy*), so terms in a user's query will literally match terms in irrelevant documents.

The proposed *latent semantic indexing* (LSI) approach tries to overcome the problems of word-based access by treating the observed word to text-object association data as an unreliable estimate of the true, larger pool of words that could have been associated with each object. It is assumed there is some underlying latent semantic structure[1] in word usage data that is partially obscured by the variability of word choice. Using the SVD defined in (1), we can estimate this latent structure and remove the obscuring *noise*.

Specifically, for an $m \times n$ *term-document* matrix A whose m rows and n columns ($m \gg n$) correspond to terms and documents, respectively, we seek the closest (in a least squares sense) rank-k ($k \ll n$) matrix

$$A_k = \sum_{i=1}^{k} u_i \cdot \sigma_i \cdot v_i^T \ \ with\ k < r\ , \tag{2}$$

given by Theorem 1.2. The idea is that the matrix A_k captures the major associational structure in the matrix and removes the noise. Since relatively few terms are used as referents to a given document, the rectangular matrix $A = [a_{ij}]$ is quite sparse. The matrix element a_{ij} indicates the frequency in which term i occurs in document j. As discussed in [13], each *raw* term frequency is usually modified using a sophisticated weighting scheme (e.g., entropy weighting) which takes into account the distribution of terms over documents. Hence, the matrix element a_{ij} may be either an integer or a rational number. Depending upon the size of the database from which the term-document is generated, the matrix A can have several thousand rows and slightly fewer columns. Table 1 lists a few statistics of ten sample sparse term-document matrices that have been generated[2]. We note that μ_r and μ_c are the average number of nonzeros per row and column, respectively. The *Density* of each sparse matrix listed in Table 1 is defined to be the ratio (Rows $\times$ Columns) / (Nonzeros).

By using the *reduced model* in (2), usually with $k = \alpha n$ ($\alpha \leq .01$), minor differences in terminology are virtually ignored. Moreover, the closeness of objects is determined by the overall pattern of term usage, so documents

[1] *Semantic structure* refers to the correlation structure in the way in which individual words appear in documents; *semantic* implies only the fact that terms in a document may be taken as referents to the document itself or to its topic.

[2] Special thanks to Sue Dumais from Bell Communications Research (Bellcore), Morristown, NJ, and Dulce Ponceleon from Apple Computer Inc., Cupertino, CA for providing all term-document matrices mentioned in this document.

Data	Source	Columns	Rows	Nonzeros	Density	μ_c	μ_r
`ADI`	Bellcore	82	374	1343	4.38	16.0	4.0
`APP1`	Apple	44	3206	7722	0.05	175.5	2.4
`APP2`	Apple	294	1472	13442	0.03	45.7	9.1
`CISI`	Bellcore	1460	5143	66340	0.88	45.4	12.9
`CRAN`	Bellcore	1400	4997	78942	1.10	56.4	15.8
`MED`	Bellcore	1033	5831	52012	0.86	50.4	8.9
`MAG`	Bellcore	425	10337	80888	1.80	190.3	7.8
`TECH`	Bellcore	6535	16637	327244	0.30	50.0	20.0
`NEWS`	Bellcore	19660	35796	1879480	0.02	95.6	50.0
`ENCY`	Bellcore	25629	56530	2843956	0.002	110.9	50.3

Table 1: Sample sparse term-document matrix specifications.

can be classified together regardless of the precise words that are used to describe them, and their description depends on a consensus of their term meanings, thus dampening the effects of polysemy. As a result, terms that do not actually appear in a document may still be used as referents, if that is consistent with the major patterns of association in the data. Position in the reduced space ($\mathrm{R}(A_k)$) then serves as a new kind of *semantic indexing*.

As discussed in [3] and [8], LSI using the sparse SVD can be more robust and economical than straight term overlap methods. However, in practice, one must compute at least 100-200 largest singular values and corresponding singular vectors of sparse matrices having similar characteristics to those matrices in Table 1. In addition, it is not necessarily the case that rank(A) = n for the $m \times n$ term-document matrix A, this is due to errors caused by term extraction, spelling, or duplication of documents. Regarding the numerical precision of the desired singular triplets for LSI, recent tests using a few of the databases listed in Table 1 have revealed that the i-th residual, $\tilde{r}_i$, corresponding to the i-th approximate singular triplet, $\{\tilde{u}_i, \tilde{\sigma}_i, \tilde{v}_i\}$, need only satisfy

$$10^{-6} \leq \|\tilde{r}_i\|_2 \leq 10^{-3},$$

where $\|\tilde{r}_i\|_2$ is defined by

$$\|\tilde{r}_i\|_2 = \left[(\|A\tilde{v}_i - \tilde{\sigma}_i\tilde{u}_i\|_2^2 + \|A^T\tilde{u}_i - \tilde{\sigma}_i\tilde{v}_i\|_2^2)^{\frac{1}{2}}\right] / \left[\|\tilde{u}_i\|_2^2 + \|\tilde{v}_i\|_2^2\right]^{\frac{1}{2}} .$$

Finally, as the desire for using LSI on larger and larger databases or archives grows, fast algorithms for computing the sparse singular value decomposition will become of paramount importance.

3 Algorithms

Before presenting algorithms for computing the sparse singular value decomposition, we note that classical methods for determining the SVD of dense matrices: the Golub-Kahan-Reinsch method ([15], [18]) and Jacobi-like SVD methods ([2], [20]) are not optimal for large sparse matrices. Since these methods apply orthogonal transformations (Householder or Givens) directly to the sparse matrix A, they incur excessive fill-in and thereby require tremendous amounts of memory. Another drawback to these methods for computing the SVD of dense matrices is that they will compute all the singular triplets of A, and hence may be computationally wasteful when only a subset of singular triplets are desired.

There are two canonical sparse symmetric eigenvalue problems which can be used to (indirectly) compute the sparse singular value decomposition. In this section, we present various iterative methods which can be applied to these sparse symmetric eigenvalue problems.

3.1 Equivalent Eigenvalue Problems

Associated with an $m \times n$ $(m \geq n)$ matrix A is the symmetric $(m+n)\times(m+n)$ matrix

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

If rank$(A) = n$, it can be easily shown that the eigenvalues of B are the n pairs, $\pm\sigma_i$, where σ_i is a singular value of A, with $(m-n)$ additional zero eigenvalues if $m > n$. The multiplicity of the zero eigenvalue of B is $m+n-2r$, where r=rank(A). The following Lemma (see [7] for proof) demonstrates how the SVD of A is generated from the eigenvalues and eigenvectors of the the matrix B in (3).

Lemma 3.1 *Let A be an $m \times n$ $(m \geq n)$ matrix and B defined by (3).*

1. *For any positive eigenvalue, σ_i, of B let $(u_i, v_i)^T$ denote a corresponding eigenvector of norm $\sqrt{2}$. Then σ_i is a singular value of A and u_i, v_i are respectively, left and right singular vectors of A corresponding to σ_i.*

2. *For $\sigma_i = 0$, if B has corresponding orthogonal eigenvectors $(u_j, v_j)^T$ with $v_j \neq 0$ and $u_j \neq 0$ for $j = 1, ...t$ for some $t \geq 1$, then 0 is a singular value of the matrix A, and the corresponding left and right singular vectors can be obtained by orthogonalizing these u_j and v_j, respectively. Otherwise, A has full rank, i.e., rank$(A) = n$.*

The numerical accuracy of the i-th approximate singular triplet $(\tilde{u}_i, \tilde{\sigma}_i, \tilde{v}_i)$ as determined via the eigensystem of the *2-cyclic*[3] matrix B (provided $A \geq 0$) is then determined by the norm of the eigenpair residual vector r_i defined as

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

which can also be written as

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

Alternatively, we may compute the SVD of A indirectly by the eigenpairs of either the $n \times n$ matrix A^TA or the $m \times m$ matrix AA^T. Lemma 3.2 illustrates the fundamental relations between these symmetric eigenvalue problems and the SVD.

Lemma 3.2 *Let A be an $m \times n$ $(m \geq n)$ matrix with rank$(A) = r$.*

1. *If $V = \{v_1, v_2, ..., v_r\}$ are linearly independent $n \times 1$ eigenvectors of A^TA so that $V^T(A^TA)V = diag(\sigma_1^2, \sigma_2^2, ..., \sigma_r^2, \underbrace{0, ..., 0}_{n-r})$, then σ_i is the i-th nonzero singular value of A corresponding to the right singular vector v_i. The corresponding left singular vector, u_i, is then obtained as $u_i = \frac{1}{\sigma_i}Av_i$.*

[3]A non-negative irreducible matrix B which is 2-cyclic has 2 eigenvalues of modulus $\rho(B)$, where $\rho(B)$ is the spectral radius of B. See Definition 2.2 on page 35 in [34].

2. *If* $U = \{u_1, u_2, ..., u_r\}$ *are linearly independent* $m \times 1$ *eigenvectors of* AA^T *so that* $U^T(AA^T)U = diag(\sigma_1^2, \sigma_2^2, ..., \sigma_r^2, \underbrace{0, ..., 0}_{m-r})$, *then* σ_i *is the* i*-th nonzero singular value of* A *corresponding to the left singular vector* u_i. *The corresponding right singular vector,* v_i, *is then obtained as* $v_i = \frac{1}{\sigma_i} A^T u_i$.

Computing the SVD of A via the eigensystems of either A^TA or AA^T may be adequate for determining several of the largest singular triplets of A, but the loss of accuracy can be severe for the smallest singular triplets (see [7]). Whereas the smallest and largest singular values of A are the extremes of the spectrum of A^TA or AA^T, the smallest singular values of A lie at the center of the spectrum of B in (3). For computed eigenpairs of A^TA and AA^T, the norms of the i-th eigenpair residuals (corresponding to (4)) are given by

$$\|r_i\|_2 = \|A^TA\tilde{v}_i - \tilde{\sigma}_i^2\tilde{v}_i\|_2 \ / \ \|\tilde{v}_i\|_2$$

and

$$\|r_i\|_2 = \|AA^T\tilde{u}_i - \tilde{\sigma}_i^2\tilde{u}_i\|_2 \ / \ \|\tilde{u}_i\|_2 \ ,$$

respectively. Thus, extremely high precision in computed eigenpairs may be necessary to compute the smallest singular triplets of A. Difficulties in approximating the smallest singular values by any of the three equivalent symmetric eigenvalue problems are discussed in [3]. The naming convention of each SVDPACKC program specfies both the algorithm and the type of equivalent eigensystem used to approximate singular triplets. Specifically, all possible entries for the three fields of the four character SVDPACKC root filename `MMTE` are given in Table 2. The contents of all binary output files are listed in Appendix C, Section 10. In the following subsections, we briefly describe the eight SVDPACKC routines which can be used to approximate the singular triplets of large sparse matrices. Details of each sparse iterative method implemented in SVDPACKC are presented in [4].

3.2 Subspace Iteration (sis1, sis2)

Subspace iteration is perhaps one of the simplest algorithms used to solve large sparse eigenvalue problems. As discussed in [26], it can be viewed as a block generalization of the classical power method. The simplest version

MMTE[.c]		
Field	Description	Possible Entries
`MM`	Method (Algorithm)	`bl` $\equiv$ Block Lanczos `la` $\equiv$ Single Vector Lanczos `si` $\equiv$ Subspace Iteration `tm` $\equiv$ Trace Minimization
`T`	File Type	`d` $\equiv$ Documentation File `p` $\equiv$ Input Parameters File `o` $\equiv$ Output File `s` $\equiv$ Source File
`E`	Eigensystem or Output Channel from SVDPACK (Fortran-77)	`1` $\equiv$ Cyclic Matrix B defined by (3) `2` $\equiv$ $A^T A$ Matrix (for comparison purposes) `2,3,8,9` $\equiv$ Output Channel

Table 2: SVDPACKC program naming convention.

of subspace iteration was introduced by Bauer ([1]) and if adapted to the matrix B in (3) would involve forming the sequence

$$Z_k = B^k Z_0 \ ,$$

where $Z_0 = [z_1, z_2, \cdots, z_s]$ is an $(m+n) \times s$. If the column vectors, z_i, are normalized separately (as done in the power method), then these vectors will converge to the dominant eigenvector of B. Thus, the matrix Z_k will progressively lose the linear independence of its columns. In order to approximate the p-largest eigenpairs of B, Bauer demonstrated that linear independence among the z_i's could be maintained if they were orthogonalized at each step, say by a modified Gram-Schmidt procedure. However, the convergence rate of the z_i's to eigenvectors of B would only be linear.

The sophisticated implementation of subspace iteration used in sis1 and sis2 is based on Rutishauser's *ritzit* program (see [28]). 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 3. The Rayleigh Quotient matrix, H_k, in step (3) is essentially the projection of 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.

The primary cost of sis1 and sis2 (as with all SVDPACKC routines) lies in the total number of sparse matrix-vector multiplications required. If $s \geq p$ vectors, z_i, are used to approximate the p-largest eigenvectors of the $(m + n) \times (m + n)$ matrix B, the cost in floating-point operations per iteration would be

$$s \times [2(1 + \mu_r)m \;+\; 2(1 + \mu_c)n] \;, \tag{5}$$

where μ_r and μ_c are the average number of nonzeros per row and column, respectively. In SVDPACKC, the multiplication of a vector by the matrices A and A^T is determined by subroutines `opa` and `opat`, respectively. Subroutine `opb` multiplies a vector by the matrix B, which may be given by (3) or $A^T A$ (with possibly a diagonal perturbation or shift).

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

Table 3: Subspace iteration as implemented in sis1 and sis2.

The orthogonal factorization in step(2) of Table 3 is computed by a modified Gram-Schmidt procedure. On multiprocessor architectures (especially those having hierarchical memories), one may achieve high performance (with a slight increase in the total number of arithmetic operations) by using either a block Gram-Schmidt or block Householder orthogonalization method in step(2). As discussed in [14], significant improvements in the algorithmic performance of fundamental linear algebra kernels may be gained through the improved data locality associated with block-based methods. For the spectral decomposition step (4), larger subspaces, an optimized implementation of the classical EISPACK ([32]) pair, TRED2 and TQL2. On parallel computers, Cuppen's algorithm as parallelized by Dongarra and Sorensen ([10]) would be effective for step (4).

3.2.1 Input Parameters

The input parameters for sis1.c and sis2.c are read from the parameter files sip1 and sip2, respectively. These files should contain the following six fields of constants and switches on a single line:

$$<name>\ em\ numextra\ km\ eps\ v$$

where

- $<name>$ is a string defining the name of the dataset.
- em is an integer specifying the number of desired triplets.
- $numextra$ is an integer specifying the number of extra vectors to carry so that the subspace dimension is $em + numextra$.
- km is an integer specifying the maximum number of iterations.
- eps is a double specifying the residual tolerance for approximated singular triplets.
- v contains the string `TRUE` or `FALSE` to indicate when singular vectors are needed (`TRUE`) and when only singular values are needed (`FALSE`).

As an example,

```
'belladit'  10   4   150   1.e-6  TRUE
```

indicates that the dataset `belladit` contains the input sparse matrix whose 10-largest singular triplets are sought to 10^{-6} accuracy using a subspace dimension of 14 for no more than 150 iterations.

3.2.2 User-Defined Routines

For all SVDPACKC programs, the actual sparse matrix is always read from a file called `matrix`. You must make sure that this file stores the sparse matrix using an appropriate format such as the Harwell-Boeing sparse matrix format [11] (see Appendix B, Section 9). All the iterative methods implemented in SVDPACKC do not modify the input matrix A, which is only referenced through matrix-vector multiplication. For subspace iteration, we provide `opb()` and `opa()` which perform different sparse matrix-vector mulitiplications. Table 4 lists these kernels, and where appropriate, their specific function in each implementation. Note that α is chosen so that B is positive definite.

code	`opb()`	`opa()`
sis1	$y = \begin{bmatrix} \alpha I & A \\ A^T & \alpha I \end{bmatrix} x$	–
sis2	$y = A^T A x$	$y = Ax$

Table 4: Matrix-vector multiplication kernels for sis1 and sis2.

In the next section, we discuss an alternative subspace method which would appear to be more suitable for multiprocessors than subspace iteration in that the desired singular triplets are iterated upon (for the most part) in parallel.

3.3 Trace Minimization Method (tms1, tms2)

Another candidate subspace method for the SVD of sparse matrices is based upon the trace minimization algorithm discussed in [30] and [37] for the generalized eigenvalue problem

$$Hx = \lambda G x \ , \tag{6}$$

where H and G are symmetric and G is also positive definite. In order to compute the SVD of an $m \times n$ matrix A, we initially replace H with $\tilde{B}$, where γ is chosen so that

$$\tilde{B} = \begin{pmatrix} \gamma I & A \\ A^T & \gamma I \end{pmatrix}, \tag{7}$$

is positive definite, or set $H = A^T A$. Since we need only consider equivalent standard symmetric eigenvalue problems (see Section 3.1), we simply define $G = I_{m+n}$ (or I_n if $H = A^T A$). Accordingly, our appropriate trace minimization SVD scheme is then based upon the following theorem which is a direct consequence of the Courant-Fischer theorem (see [35]). Without loss of generality, let us assume that $H = \tilde{B}$, $G = I_{m+n}$ and consider the associated symmetric eigensystem of order $m + n$ (tms1).

Theorem 3.1 *Let $\tilde{B}$ be as given in (7) and let $\mathcal{Y}$ be the set of all $(m+n) \times p$ matrices Y for which $Y^T Y = I_p$. Then*

$$\min_{Y \in \mathcal{Y}} \mathrm{trace}(Y^T \tilde{B} Y) = p\gamma - \sum_{i=1}^{p} \sigma_i \ ,$$

where σ_i is a singular value of A, $\lambda_i = \gamma \pm \sigma_i$ is an eigenvalue of $\tilde{B}$, and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p$.

Given an $(m + n) \times p$ matrix Y which forms a *section* of the eigenvalue problem

$$\tilde{B} z = \lambda z \ , \tag{8}$$

i.e.,

$$Y^T \tilde{B} Y = \tilde{\Sigma} \ , Y^T Y = I_p \ , \tag{9}$$
$$\tilde{\Sigma} = \mathrm{diag}(\tilde{\sigma}_1, \tilde{\sigma}_2, \cdots, \tilde{\sigma}_p) \ ,$$

tms1 finds a sequence of iterates $Y_{k+1} = F(Y_k)$, where both Y_k and Y_{k+1} form a section of (8), and have the property $\mathrm{trace}(Y_{k+1}^T \tilde{B} Y_{k+1}) < \mathrm{trace}(Y_k^T \tilde{B} Y_k)$. From Theorem 3.1, the matrix Y in (9) which minimizes $\mathrm{trace}(Y^T \tilde{B} Y)$ is the matrix of $\tilde{B}$-eigenvectors associated with the p-smallest eigenvalues of the problem (8). As discussed in [30] and [37], $F(Y)$ can be chosen so that global convergence is assured. Moreover, (8) can be regarded as the quadratic minimization problem

$$\text{minimize } \mathrm{trace}(Y^T \tilde{B} Y) \tag{10}$$

subject to the constraints

$$Y^T Y = I_p \ . \tag{11}$$

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

$$\begin{pmatrix} \tilde{B} & Y_k \\ Y_k^T & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L \end{pmatrix} = \begin{pmatrix} \tilde{B}Y_k \\ 0 \end{pmatrix} , \tag{12}$$

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

Since the matrix $\tilde{B}$ is positive definite (by construction), one can alternatively consider the independent (parallel) subproblems

$$\text{minimize trace}(\ (y_j^{(k)} - d_j^{(k)})^T \tilde{B}(y_j^{(k)} - d_j^{(k)})\ ) \tag{13}$$

subject to the constraints

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

where $d_j^{(k)} = \Delta_k e_j$, e_j is a vector composed of all zeros except for the value 1 in the j-th component, 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 (11), i.e., so that (see [22])

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

We then recast (12) as

$$\begin{pmatrix} \tilde{B} & Y_k \\ Y_k^T & 0 \end{pmatrix} \begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix} = \begin{pmatrix} \tilde{B}y_j^{(k)} \\ 0 \end{pmatrix} \ , j = 1, 2, \ldots, p, \tag{14}$$

where $2l$ is a vector of order p reflecting the Lagrange multipliers.

The solution of the p systems of linear equations in (14) 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 use an iterative method (conjugate gradient) within tms1 and tms2.

3.3.1 Polynomial Acceleration Technique (tms2)

The Chebyshev acceleration strategy used within subspace iteration (see Section 3.2) is also used in tms2. However, to dampen unwanted singular values of A in this context we must solve the generalized eigenvalue problem (as opposed to (8))

$$x = \frac{1}{P_q(\lambda)} P_q(A^T A)x \ , \tag{15}$$

where $P_q(x) = T_q(x) + \epsilon I_{m+n}$, $T_q(x)$ is the Chebyshev polynomial of degree q, and ϵ is chosen so that $P_q(A^T A)$ is (symmetric) positive definite. The appropriate quadratic minimization problem similar to (13) for (15) can be expressed as

$$\text{minimize trace}(\ (y_j^{(k)} - d_j^{(k)})^T (y_j^{(k)} - d_j^{(k)})\ ) \tag{16}$$

subject to the constraints

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

In effect, we then approximate the smallest singular values of A (or eigenvalues of $\tilde{B}$) as the *largest* eigenvalues of the matrix $P_q(A^T A)$ whose gaps are considerably larger than those of the eigenvalues of $A^T A$.

Although the additional number of sparse matrix-vector multiplications associated with the multiplication by $P_q(A^T A)$ will be significant for high degrees q, the system of equations via Lagrange multipliers in (14) becomes much easier to solve, i.e.,

$$\begin{pmatrix} I & P_q(A^T A) Y_k \\ Y_k^T P_q(A^T A) & 0 \end{pmatrix} \begin{pmatrix} d_j^{(k)} \\ l \end{pmatrix} = \begin{pmatrix} y_j^{(k)} \\ 0 \end{pmatrix} \ , j = 1, 2, \ldots, p. \tag{17}$$

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

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

Thus, we need not employ the use of an iterative solver for determining Y_{k+1} since the matrix $\left[Y_k^T P_q^2(\tilde{B}) Y_k \right]^{-1}$ is of order p and using the orthogonal factorization

$$P_q(A^T A) Y_k = \hat{Q} \hat{R} \ ,$$

we have

$$\left[Y_k^T P_q^2(A^T A) Y_k\right]^{-1} = \hat{R}^{-T} \hat{R}^{-1} \ .$$

The control of the polynomial degree, q, is determined by the strategy discussed in [28] and [3] for damping the unwanted singular values of A which correspond to a particular choice for H in (6). We note that tms2 with the choice $H = A^T A$ can be used to approximate the p-smallest singular values of A, whereas the choice $H = \gamma^2 I - A^T A$ (γ any least upper bound for σ_{max}) will enable tms2 to approximate the p-largest singular values of A. tms1, which determines the eigensystem of the matrix $\tilde{B}$ in (7), does not currently employ polynomial acceleration.

3.3.2 Shifting Strategy (tms1, tms2)

As discussed in [30], we can also accelerate the convergence of the Y_k to eigenvectors of $\tilde{B}$ (and hence singular vectors of A) by incorporating Ritz shifts (see [26]) into both tms1 and tms2. Specifically, we modify the symmetric eigenvalue problem in (8) as

$$(\tilde{B} - \nu_j^{(k)} I) z_j = (\lambda_j - \nu_j^{(k)}) z_j \ , j = 1, 2, \ldots, s \ , \tag{18}$$

where $\nu_j^{(k)} = \tilde{\sigma}_j^{(k)}$ is the j-th approximate eigenvalue (9) from the k-th TRSVD iteration, and λ_j, z_j are an exact eigenpair of $\tilde{B}$. In other words, we simply use our most recent approximations to the eigenvalues of $\tilde{B}$ from our k-th iteration as Ritz shifts. As was shown by Wilkinson in [36], the Rayleigh quotient iteration associated with (18) will ultimately achieve cubic convergence to $\gamma - \sigma_j$, where σ_j is an exact singular value of A, provided $\nu_j^{(k)}$ is sufficiently close to $\gamma - \sigma_j$. However, since we have $\nu_j^{(k+1)} < \nu_j^{(k)}$ for all k (see Theorem 3.1), i.e., we approximate eigenvalues of $\tilde{B}$ from above, $\tilde{B} - \nu_j^{(k)} I$ will not be positive definite and thus we cannot guarantee the convergence of this shifted method for any particular singular triplet j. However, the strategy outlined in [4] has been quite successful in maintaining global convergence with shifting.

The logic of tms2 which appropriately utilizes polynomial (Chebyshev) acceleration prior to Ritz shifting is outlined in [4]. It is important to note that once shifting has been invoked (Step (4)) tms2 abandons the use of Chebyshev polynomials $P_q(\tilde{B})$ and solves shifted systems ($\tilde{B}$ replaced by $\tilde{B} -$

$\nu_j^{(k)} I)$ of the form in (12). The *context switch* from either non-accelerated or polynomial-accelerated trace minimization iterations to trace minimization iterations with Ritz shifting is accomplished by monitoring the reduction of the residuals (4) for isolated eigenvalues or clusters of eigenvalues (see [4]).

For an isolated eigenvalue approximation $\tilde{\sigma}_j^{(k)}$, which is detected say after k_0 tms2 iterations, we monitor succeeding iterations ($k > k_0$), and determine if the norm of the current residual ($\|r_j^{(k)}\|_2$) is less than a chosen order of magnitude ($\eta = 10^{-t}$, for a small integer t) of $\|r_j^{(k_0)}\|_2$. Thus, the parameter η serves as our control for the context switch from polynomial-based acceleration to shift-based acceleraton. The value of η will naturally depend upon the desired accuracy of the singular triplets sought, since we would like to produce suitable shifts ($\nu_k^j \equiv \tilde{\sigma}_k^j$) for fast convergence of y_k^j to an eigenvector of $\tilde{B}$. For most problems considered thus far, optimal convergence rates can be obtained with $\eta = 10^{-1}, 10^0$.

3.3.3 Input Parameters

The input parameters for tms1.c and tms2.c are read from the parameter files tmp1 and tmp2, respectively. These files should contain the following eight fields of constants and switches on a single line:

$$<name>\ p\ s\ job\ tol\ red\ v\ maxi$$

where

- *<name>* is a string defining the name of the dataset.
- p is an integer specifying the number of desired triplets.
- s is an integer specifying the dimension of the subspace to use.
- *job* is an integer specifying the type of acceleraton to be used.

  $job = 0$: No acceleration strategy used.

  $job = 1$: Ritz-shifting used.

  $job = 2$: Chebyshev polynomials and Ritz-shifting used.
- *tol* is a double specifying the residual tolerance for approximated singular triplets.

- *red* is a double specifying the residual reduction factor to initiate Ritz-shifting (when $job = 1, 2$).
- v contains the string `TRUE` or `FALSE` to indicate when singular vectors are needed (`TRUE`) and when only singular values are needed (`FALSE`).
- *maxi* is an integer specifying the maximum number of iterations.

As an example,

```
'belladit'  10   12     1   1.e-6   1.0e0   TRUE    80
```

indicates that the dataset `belladit` contains the input sparse matrix whose 10-largest singular triplets are sought to 10^{-6} accuracy using a subspace dimension of 12 for no more than 80 trace minimization iterations. Ritz-shifting acceleration is used and all residual errors for clustered or isolated approximate singular values must be a factor of 1.0e0 smaller than their initial residual errors prior to Ritz-shifting.

3.3.4 User-Defined Routines

For trace minimization, we provide `opb()` and `opat()` which perform the sparse matrix-vector mulitiplications listed in Table 5.

code	`opb()`	`opat()`
tms1	$y = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} x$	$y = A^T x$
tms2	$y = A^T A x$	$y = A^T x$

Table 5: Matrix-vector multiplication kernels for tms1 and tms2.

3.4 Single-Vector Lanczos Method (las1, las2)

Other popular methods for solving large, sparse, symmetric eigenproblems originate 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 original matrix B's extremal eigenvalues. Suppose we consider the $(m+n) \times (m+n)$ 2-cyclic matrix B given in (3), where A is the $m \times n$ matrix whose singular triplets are sought, and let v_1 be a randomly generated starting $(m+n) \times 1$ vector such that $\|v_1\|_2 = 1$. For $j = 1, 2, ..., l$ define the corresponding Lanczos matrices T_j using the following recursion ([25]). Define $\beta_1 \equiv 0$, and $v_0 \equiv 0$. Then, for $i = 1, 2, ..., l$ define Lanczos vectors w_i and scalars α_i and β_{i+1} where

$$\beta_{i+1} w_{i+1} = B w_i - \alpha_i w_i - \beta_i w_{i-1}\ , \text{ and} \tag{19}$$

$$\alpha_i = w_i^T (B w_i - \beta_i w_{i-1})$$

$$|\beta_{i+1}| = \|B w_i - \alpha_i w_i - \beta_i w_{i-1}\|_2\ .$$

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

$$T_j \equiv \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \beta_j \\ & & & & \beta_j & \alpha_j \end{pmatrix} . \tag{20}$$

By definition, the vectors $\alpha_i w_i$ and $\beta_i w_{i-1}$ in (19) are respectively, the orthogonal projections of $B w_i$ onto the most recent w_i and w_{i-1}. Hence for each i, the next Lanczos vector w_{i+1} is obtained by orthogonalizing $B w_i$ with respect to w_i and w_{i-1}. The resulting α_i, β_{i+1} obtained in these orthogonalizations define the corresponding Lanczos matrices. If we rewrite (19) in matrix form, then for each j we have

$$B W_j = W_j T_j + \beta_{j+1} w_{j+1} e_j^T\ , \tag{21}$$

where $W_j \equiv [w_1, w_2, ..., w_j]$ is the $n \times j$ matrix whose k-th column is the k-th Lanczos vector, and e_j^T is the j-th column of the $n \times n$ identity matrix. Thus, the Lanczos recursion (21) generates a family of real symmetric tridiagonal matrices related to both B and w_1. Table 6 outlines the basic Lanczos procedure for computing the eigenvalues and eigenvectors of the symmetric 2-cyclic matrix B.

(1)	Use any variant of the Lanczos recursion (19) to generate a family of real symmetric tridiagonal matrices, T_j $(j = 1, 2, ..., q)$.
(2)	For some $k \leq q$, compute relevant eigenvalues of T_k.
(3)	Select some or all of these eigenvalues as approximations to the eigenvalues of the matrix B, and hence singular values of A.
(4)	For each eigenvalue λ compute a corresponding unit eigenvector z such that $T_k z = \lambda z$. Map such vectors into corresponding Ritz vectors $y \equiv W_q z$, which are then used as approximations to the desired eigenvectors (singular vectors) of the matrix B (A).

Table 6: Single-vector Lanczos recursion used in las1 and las2.

As with the previous SVDPACKC methods, the matrix B is only referenced through matrix-vector multiplication in Table 6. At each iteration, the basic Lanczos recursion requires only the two most recently-generated vectors, although for finite-precision arithmetic modifications suggested by Parlett and Scott [27], and Simon [31] require additional Lanczos vectors to be readily accessible via secondary storage.

In one sense, the Lanczos procedure can be viewed as the Gram-Schmidt orthogonalization of the set of Krylov vectors $w_1, Bw_1, ..., B^{k-1}w_1$. Alternatively, span $\{W_j\}$ is a Krylov subspace for the matrix B, and the Lanczos procedure is a mechanism for generating orthonormal bases for these Krylov subspaces and for computing the orthogonal projection of B onto these subspaces. Computing the eigenvalues of the T_j's is equivalent to computing the best approximations to the eigenvalues and eigenvectors of B restricted to the corresponding Krylov subspaces. The accuracy of these approximations have been studied in detail by Kaniel ([21]) and more recently by Saad ([29]). Saad improved the original error bounds of Kaniel, however, but these results still indicate deterioration of accuracy of the computed eigenvalues and of the corresponding Ritz vectors as we move to the interior of the spectrum of B.

In using finite-precision arithmetic, any practical Lanczos procedure must address problems created by losses in the orthogonality of the Lanczos vectors, w_i. Such problems include the occurrence of numerically-multiple eigenvalues of T_j (for large j) for simple eigenvalues of B, and the appearance of spurious eigenvalues among the computed eigenvalues for some T_j. Approaches to deal with these problems range from two different extremes. The total reorthogonalization of every Lanczos vector with respect to every previously-generated Lanczos vector is one extreme ([16]). The other approach accepts the loss in orthogonality and then deals with these problems directly. Regarding storage requirements, supercomputers such as the CRAY-2S/4-128 with 128 megawords (1024 million bytes) of core memory may be sufficient for most Lanczos recursions requiring total reorthogonalization (for $m \times n$ matrices in which $mn \ll 10^7$). 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 computer storage requirements. Such a procedure must track (see [7]) the spurious eigenvalues of B (singular values of A) associated with the loss of orthogonality in the Lanczos vectors, w_i.

las1 and las2 implement a single-vector Lanczos algorithm (19) equipped with a selective reorthogonalization strategy. Both programs evolved from the LANSO program (Version 1, April 1989) designed by Parlett and his colleagues at The University of California at Berkeley ([27], [31]). The original LANSO program was primarily designed for the standard and generalized symmetric eigenvalue problem. las1 and las2are adaptations of LANSO when eigensystems of the 2-cyclic matrix B defined in (3) and $B = A^T A$, respectively, are desired.

3.4.1 Input Parameters

The input parameters for las1.c and las2.c are read from the parameter files lap1 and lap2, respectively. These files should contain the following seven fields of constants and switches on a single line:

<name> lanmax maxprs endl endr vectors kappa

where

- <*name*> is a string defining the name of the dataset.
- *lanmax* is an integer specifying the maximum number of Lanczos iterations allowed.
- *maxprs* is an integer which indicates the number of singular triplets of A (eigenpairs of the equivalent matrix B) desired.
- *endl,endr* are integers specifying the two end-points of an interval within which all unwanted eigenvalues of the particular matrix B lie.
- *vectors* contains the string `TRUE` or `FALSE` to indicate when singular triplets are needed (`TRUE`) and when only singular values are needed (`FALSE`).
- *kappa* is a double containing the relative accuracy of Ritz values acceptable as eigenvalues of the matrix B.

As an example,

```
'belladit'  50  10  -1.e-30  1.e-30  TRUE  1.0e-6
```

indicates that the dataset `belladit` contains the input sparse matrix whose 10-largest singular triplets are sought to 10^{-6} accuracy using a Krylov subspace of dimension less than or equal to 50. Here we also suppress the extraction of (computationally) zero Ritz values in the interval $[-10^{-30},\ 10^{-30}]$.

3.4.2 User-Defined Routines

For our single-vector Lanczos programs, we provide `opb()` and `opa()` which perform the sparse matrix-vector mulitiplications listed in Table 7.

3.5 Block Lanczos Method (bls1, bls2)

As subspace iteration is the block generalization of the classical power method for computing eigenpairs, we now consider a block analogue of the single vector Lanczos recursion given in (19). Exploiting the structure of the matrix B in (3), the following Lemma presents an alternative form for the Lanczos recursion (19).

code	opb()	opa()
las1	$y = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} x$	–
las2	$y = A^T A x$	$y = Ax$

Table 7: Matrix-vector multiplication kernels for las1 and las2.

Lemma 3.3 *Let A be an $m \times n$ $(m \geq n)$ matrix and B defined by (3). Apply the Lanczos recursion specified by (19) to B with a starting vector $\tilde{u} = (u, 0)^T$ such that $\|\tilde{u}\|_2 = 1$. Then the diagonal entries of the real symmetric tridiagonal Lanczos matrices generated are all identically zero, and the Lanczos recursion in (19) reduces to the following. Define $u_1 \equiv u$, $v_0 \equiv 0$, and $\beta_1 \equiv 0$. For $i = 1, 2, ..., k$ we then obtain the Lanczos recursions*

$$\begin{aligned} \beta_{2i} v_i &= A^T u_i - \beta_{2i-1} v_{i-1} \ , \\ \beta_{2i+1} u_{i+1} &= A v_i - \beta_{2i} u_i \ . \end{aligned} \tag{22}$$

The Lanczos recursion (22), 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 ([33], [19]), (22) may be represented in matrix form as

$$\begin{aligned} A^T \hat{U}_k &= \hat{V}_k J_k^T + Z_k \ , \\ A \hat{V}_k &= \hat{U}_k J_k \ + \tilde{Z}_k, \end{aligned} \tag{23}$$

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

$$K_k \equiv \begin{pmatrix} O & J_k \\ J_k^T & O \end{pmatrix} \ . \tag{24}$$

For the block analogue of (23), we make the simple substitutions

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

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

$$J_k \equiv \begin{pmatrix} S_1 & R_1^T & & & \\ & S_2 & R_2^T & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ & & & & \cdot \quad R_{k-1}^T \\ & & & & S_k \end{pmatrix} , \qquad (25)$$

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. In essence, the 2-cyclic matrix K_k in (24) is the projection of the vectors defined by

$$\begin{pmatrix} \hat{U}_k & O \\ O & \hat{V}_k \end{pmatrix}$$

onto the Krylov subspace generated by the 2-cyclic matrix B in (3). 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 in $\hat{U}_k$ and $\hat{V}_k$. Specifically, if $\{\sigma_i^{(k)}, y_i^{(k)}, z_i^{(k)}\}$ is the i-th singular triplet of J_k, then the approximation to the i-th singular triplet of A is given by $\{\sigma_i^{(k)}, \hat{U}_k y_i^{(k)}, \hat{V}_k z_i^{(k)}\}$, where $\hat{U}_k y_i^{(k)}$, $\hat{V}_k z_i^{(k)}$ are the left and right approximate singular vectors, respectively. The computation of singular triplets for J_k requires two phases. The first phase reduces J_k to bidiagonal form, say B_k, where

$$B_k \equiv \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ & & & & \cdot \quad \beta_{bk-1} \\ & & & & \alpha_{bk} \end{pmatrix} , \qquad (26)$$

via a finite sequence of orthogonal transformations (thus preserving the singular values of J_k). The second phase reduces B_k to diagonal form by a modified QR algorithm. This diagonalization procedure is discussed in detail in [18]. The resulting diagonalized B_k will yield the approximate singular values of A and the corresponding left and right singular vectors are determined by products of all the left and right transformations (respectively) used in both phases of the SVD of J_k.

There are a few options for the reduction of J_k to the bidiagonal matrix, B_k. Golub, Luk, and Overton in [17] advocated the use of either band Householder or band Givens methods which in effect *chase off* (or zero) elements on the diagonals above the first super-diagonal of J_k. We note that the algorithm for diagonalizing B_k typically requires in excess of $8(bk)^3$ multiplications (with standard Givens rotations) and thus may dominate any savings in the reduction to bidiagonal form. In either reduction (bi-diagonalization or diagonalization), the computations are primarily sequential and offer limited data locality or parallelism for possible exploitation on a multiprocessor architecture. For this reason, we adopt the single vector Lanczos bi-diagonalization recursion by (22) and (23) in bls1 for reducing the upper block bidiagonal matrix J_k to bidiagonal form (B_k), i.e.,

$$\begin{aligned} J_k^T \hat{Q} &= \hat{P} B_k^T \ , \\ J_k \hat{P} &= \hat{Q} B_k \ , \end{aligned} \tag{27}$$

or

$$\begin{aligned} J_k p_j &= \alpha_j q_j + \beta_{j-1} q_{j-1} \ , \\ J_k^T q_j &= \alpha_j p_j + \beta_j p_{j+1} \ , \end{aligned} \tag{28}$$

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 or order $bk \times bk$ and the α_i's and β_i's are defined by (26). The recursions in (28) require band matrix-vector multiplications which can be easily exploited by optimized level-2 BLAS routines ([9]) now resident in optimized mathematical libraries on most high-performance computers. For orthogonalization of the outermost Lanczos vectors, $\{U_i\}$ and $\{V_i\}$, as well as the innermost Lanczos vectors, $\{p_i\}$ and $\{q_i\}$, we have chosen to apply a complete or total reorthogonalization ([16]) strategy to insure robustness in our triplet approximations for the matrix A.

As an alternative to the outer recursion in Table 8, which is derived from the equivalent eigenvalue in the 2-cyclic matrix B, Table 10 depicts the simplified outer block Lanczos recursion for approximating the eigensystem of $A^T A$ (bls2). Combining the equations in (23), we obtain

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

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

$$H_k \equiv \begin{pmatrix} S_1 & R_1^T & & & & \\ R_1 & S_2 & R_2^T & & & \\ & R_2 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & R_{k-1}^T \\ & & & & R_{k-1} & S_k \end{pmatrix}, \tag{29}$$

having block size b. bls2 applies the block Lanczos recursion ([16]) in Table 10 for computing the eigenpairs of the $n \times n$ symmetric positive definite matrix $A^T A$. The tridiagonalization of H_k via an inner Lanczos recursion follows from simple modifications to Table 9. Analogous to the diagonalization of B_k in (25), the computation of eigenpairs of the resulting tridiagonal matrix in this case can be performed via a QR-based symmetric eigensolver.

(1)	[Formation of J_k]
	Choose V_1 ($n \times b$ and orthonormal) and $c = max\ \{bk\}$. Compute $W_1 = AV_1$. ($P_0 = U_0 = 0$ initially) Orthogonalize W_1 against U_0 (i.e., $W_1 = (I - U_0 U_0^T) W_1$).
	For $i = 2, 3, \ldots, k$ do: ($k = \lfloor c/b \rfloor$)
(1a)	Compute $Y_i = A^T U_{i-1} - V_{i-1} S_{i-1}$,
(1b)	Orthogonalize Y_i against $\{V_l\}_{l=0}^{i-1}$,
(1c)	Factor $Y_i = V_i R_{i-1}$,
(1d)	Compute $W_i = AV_i - U_{i-1} R_{i-1}^T$,
(1e)	Orthogonalize W_i against $\{U_l\}_{l=0}^{i-1}$,
(1f)	Factor $W_i = U_i S_i$.

Table 8: Hybrid Lanczos outer iteration used in bls1.

The conservation of computer memory for bls1 and bls2is insured by enforcing an upper bound, c, for the order (bk) of any J_k constructed (see Table

(2)	[Bidiagonalization of J_k, Formation of B_k]
	Choose p_1 ($\|p_1\|_2 = 1$),
	Compute $t_1 = J_k p_1$, $\alpha_1 = \|t_1\|_2$,
	For $i = 1, 2, \ldots, \tilde{n}$ do: ($\tilde{n} = b \times k$)
	(while $\alpha_j \neq 0$) do:
	$q_j = t_j / \alpha_j$,
(2a)	Compute $z_j = J_k^T q_j - \alpha_j p_j$,
(2b)	Orthogonalize z_j against $\{p_l\}_{l=1}^{j}$,
	$\beta_j = \|z_j\|_2$,
	(while $\beta_j \neq 0$) do:
	$p_{j+1} = z_j / \beta_j$,
(2c)	Compute $t_{j+1} = J_k p_{j+1} - \beta_j q_j$,
(2d)	Orthogonalize t_{j+1} against $\{q_l\}_{l=1}^{j}$,
	$\alpha_{j+1} = \|t_{j+1}\|_2$.

Table 9: Hybrid Lanczos inner iteration used in bls1.

8). This technique was suggested by Golub, Luk, and Overton in [17], and by Cullum and Donath in [6]. Given a block size b (usually $b \leq p$, where p is the desired number of triplets), the number of diagonal blocks, d, for J_k, is defined as $\lfloor c/b \rfloor$, where $\lfloor \cdot \rfloor$ denotes truncation of the mantissa. If $d < 2$, one may reset $b = c/2$ and then redefine $d = \lfloor c/b \rfloor$ so that J_k maintains the block upper bidiagonal form in (25).

As mentioned above, we may compute the SVD of the bidiagonal matrix B_k by a modified QR algorithm. Using (27) we may write

$$B_k = \bar{Q} \Sigma \bar{P}^T \ ,$$

so that

$$J_k = \hat{Q} \bar{Q} \Sigma \bar{P}^T \hat{P}^T \ ,$$

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

Table 10: Hybrid Lanczos outer iteration used in bls2.

where Σ = diag $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, and σ_i is an approximation to an exact singular value of the original $m \times n$ sparse matrix A. Hence, via (23) approximations to the i-th left and right singular vectors corresponding to σ_i are given by

$$\begin{aligned} \bar{u}_i &= \hat{U}_k \hat{Q} \bar{q}_i \ , \\ \bar{v}_i &= \hat{V}_k \hat{P} \bar{p}_i \ , \end{aligned} \tag{30}$$

where $\bar{p}_i$, $\bar{q}_i$ are the i-th columns of $\bar{P}$, $\bar{Q}$, respectively. Suppose that before restarting the outer iteration in Table 8 we have determined that p_0 singular triplets are acceptable to a user-supplied tolerance for the residual error defined in (4). Then, we update the values of the block size (b), the maximum allowable order for J_k (c), the number of diagonal blocks for J_k (d), and the number of triplets yet to be found (p) as follows:

$$\begin{aligned} b_{new} &= b_{old} - p_0 \ , \text{ if } b \geq p_{old}, \\ &= min\ \{b_{old}\ ,\ p_{old} - p_0\} \ \text{ otherwise}, \\ c_{new} &= c_{old} - p_0 \ , \\ p_{new} &= p_{old} - p_0 \ , \\ d_{new} &= \lfloor c_{new} / b_{new} \rfloor \ . \end{aligned} \tag{31}$$

All converged left and right singular vector approximations are respectively stored in matrices U_0 and V_0 so that

$$\begin{aligned} U_0 &\equiv (U_0|\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_{p_0}) \ , \\ V_0 &\equiv (V_0|\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{p_0}) \ , \end{aligned}$$

where $U_0 = V_0 = 0$ initially (prior to any restart). To estimate the accuracy of our approximate singular triplets in say iteration l, we may conveniently estimate the residual (4) for some σ_k by $\|y_k\|_2$ of Step (1a) in Table 8 for iteration $l+1$, where y_k is the k-th column of the $n \times b$ matrix Y_i. Hence, at the start of iteration $l+1$ we can determine the accuracy of our approximations from iteration l.

As with the other SVDPACKC methods, bls1 and bls2 only access the sparse matrices A and A^T through sparse matrix-vector multiplications. Some efficiency, however, is gained in the outer (block) Lanczos iterations by the multiplication of b vectors (Steps (1a), (1b) in Table 8) rather than by a single vector. These dense vectors may be stored in a fast local memory (cache) of any hierarchical memory-based architecture, and thus yield more effective data reuse. The total reorthogonalization strategy and deflation of converged singular vector approximations is accomplished in Steps (1b), (1e) in Table 8 and Steps (2b), (2d) in Table 9. A stable variant of Gram-Schmidt orthogonalization ([28]), which requires efficient dense matrix-vector muliplication (level-2 BLAS) routines ([9]), 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

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

The convergence of the block Lanczos recursion (23) in the approximation of the b-largest singular values of the matrix A is analyzed in [3]. Although the bound in [3] is somewhat tighter than that which was considered by Underwood in [33] for the symmetric eigenvalue problem, both results clearly indicate the desire for b (block size) to be chosen so that $\sigma_i - \sigma_{i+b}$ is as large as possible.

3.5.1 Input Parameters

The input parameters for bls1.c and bls2.c are read from the parameter files blp1 and blp2, respectively. These files should contain the following seven fields of constants and switches on a single line:

$$<name>\ maxit\ nc\ nb\ nums\ tol\ vtf$$

where

- $<name>$ is a string defining the name of the dataset.
- *maxit* is an integer specifying the maximum number of (outer) block Lanczos iterations allowed.
- *nc* is an integer specifying the upper bound for the Krylov subspace generated via the outer iteration.
- *nb* is an integer specifying the initial block size for the outer iteration.
- *nums* is an integer specifying the number of singular triplets desired.
- *tol* is a double specifying the residual tolerance for approximated singular triplets.
- *vtf* contains the string `TRUE` or `FALSE` to indicate when singular vectors are needed (`TRUE`) and when only singular values are needed (`FALSE`).

As an example,

```
'belladit'   40   60   4   10   1.0e-6   TRUE
```

indicates that the dataset `belladit` contains the input sparse matrix whose 10-largest singular triplets are sought to 10^{-6} accuracy using a maximum Krylov subspace dimension of 60 for no more than 40 iterations. The initial block size to be used is 4. In general, the initial block size should at least be as large as the greatest multiplicity of any singular value of A.

3.5.2 User-Defined Routines

For our block Lanczos programs, we provide `opb()`, `opa()`, and `opat()` which perform the sparse matrix-vector mulitiplications listed in Table 11. Note that the `opm()` routine performs multiplications of the appropriate matrix B times a block of dense vectors (X).

code	opb()	opm()	opa()	opat()
bls1	–	–	$y = Ax$	$y = A^T x$
bls2	$y = A^T Ax$	$Y = A^T AX$	$y = Ax$	–

Table 11: Matrix-vector multiplication kernels for bls1 and bls2.

4 SVDPACKC Interface

Before presenting our SVDPACKC benchmarks in Section 5, we illustrate how a simple yet effective interface allows users to easily generate a series of experiments using any or all of the SVDPACKC codes. Using the UNIX[4]pattern scanning and processing language, *awk*, and stream editor, *sed*, two scripts (`svdrun`, `svdsum`) for executing and tabulating the output of SVDPACKC programs have been developed. Equivalent versions of `svdrun` and `svdsum` for release 3.3 of the Macintosh Programmer's Workshop (MPW) environment on the Macintosh II/fx are also available.

`svdrun` is designed to aid in the selection of parameters (see Section 4.2) for each method, and `svdsum` produces `<input file>.sumn` files (where `n` is an integer) of tabulated output data for simplified performance comparisons. `svdrun` reads a tabulated set of parameters from a user-specified input file, and invokes the corresponding SVDPACKC routines. For the input file `svdin`, `svdsum` generates `svdin.sumn` files from the output files produced by the individual SVDPACKC runs. As illustrated by the sample `svdrun` input file, `svdin`, in Figure 1, a user can specify a sequence of experiments in order to (*i.*) compare the performance of different algorithms on one or more datasets, or (*ii.*) determine the effects of parameter choices for a particular algorithm.

Figures 1 through 3 illustrate (*i.*) and (*ii.*) for the term-document matrix, `BELLADIT`, provided in the SVDPACKC software distribution package.

[4]UNIX is a trademark of AT&T Bell Laboratories.

Generating several summary files similar to `svdin.sum1` and `svdin.sum2` in Figures 2 and 3, respectively, not only allows the user to observe trends in SVDPACKC performance across machines and/or datasets, but also creates formal benchmark characterizations for future reference and comparison. `svdrun` is portable to any UNIX-based programming environment with only modifications (related to the Fortran compiler and its options), and the `svdsum` script processes all output files residing in a current or remote working directory.

5 SVDPACKC Workstation Benchmarks

In this section, we present sample SVDPACKC benchmarks on workstations such as the Macintosh II/fx and Sun-4/490. Model SVD problems using the sparse matrix test suite defined in Table 12 are solved. These benchmarks illustrate the typical elapsed user CPU time expired by the 8 SVDPACKC programs when computing several of the largest singular triplets of *real* sparse matrices arising from applications such as information retrieval. For all the experiments reported, we use 64-bit arithmetic and seek triplets whose residuals (4) are no larger than 10^{-6}. On both the Macintosh II/fx and Sun-4/490, we use the `svdrun` script (see Section 4) to execute each SVDPACKC program on the test suite in Table 12.

5.1 Sparse Matrix Test Suite

The 29 matrices listed in Table 12, which arise from information retrieval and linear programming applications, were obtained from Apple Computer Inc., Cupertino. CA. The first 13 datasets (`APPLE1` through `WMURRAYC2`) are term-document matrices which can be used for information retrieval applications (see Section 2). The 16 remaining sparse rectangular matrices were extracted from a set of linear programming test problems compiled at Stanford University [23]. From Table 12, we can see that all of these matrices are less than 1% dense. We note that μ_r and μ_c are the average number of nonzeros per row and column, respectively. The *Density* of each sparse matrix listed in Table 12 is defined to be the ratio (Rows $\times$ Columns) / (Nonzeros).

```
#------------------------------------------------------------
# Input file used for SVDRUN script
#------------------------------------------------------------
#
# DESCRIPTION OF FIELDS:
#------------------------------------------------------------
#
# NUM --> Run number (1,2,3,...).
#
# CDE --> SVDPACKC code (i.e. las1, bls1, las2, bls2, etc.).
#
# FNM --> Sparse Matrix datafile (i.e. app1, app2, etc.).
#
# MXI --> Maximum number of iterations for method.
#
# TRP --> Number of singular triplets desired.
#
# SUB --> Maximum subspace dimension.
#
# BSZ --> Initial blocksize size (if applicable).
#
# ACC --> Accuracy (residual tolerance)
#
# VEC --> Compute singular vectors also? (TRUE/FALSE)
#------------------------------------------------------------
#
#  NUM CDE  FNM    MXI  TRP  SUB  BSZ   ACC      VEC
#  --- ---  ---    ---  ---  ---  ---   ---      ---
   1   las1 app1   200   10              1.0e-6   TRUE
   2   las2 app1    44   10              1.0e-6   TRUE
   3   bls1 app1   200   10   40    2    1.0e-6   TRUE
   4   bls2 app1   200   10   40    4    1.0e-6   TRUE
   5   las1 app2   200   10              1.0e-6   TRUE
   6   las2 app2   200   10              1.0e-6   TRUE
   7   bls1 app2   200   10   40    2    1.0e-6   TRUE
#------------------------------------------------------------
```

Figure 1: Sample input file, `svdin`, used by `svdrun` script for SVDPACKC user interface.

PROGRAM	las1	las2	bls1	bls2	sis1
DATASETS	'belladit'	'belladit'	'belladit'	'belladit'	'belladit'
FILENAME	belladit.out1	belladit.out2	belladit.out3	belladit.out4	belladit.out5
DATE	Feb 19 1993	Feb 19 1993	Feb 19 1993	Feb 19 1993	Feb 19 1993
MAX. NO. OF ITERATIONS	50	44	80	40	150
ORDER OF EIGENSYSTEM	456	82	456	82	456
ROW DIMENSION OF A	374	374	374	374	374
COLUMN DIMENSION OF A	82	82	82	82	82
AUXILLARY MEMORY(BYTES)	4.47+05	2.34+05	622088	168688	3354784
WANT S-VECTORS? [T/F]	T	T	T	T	T
NO. OF STEPS/ITERATIONS	50	44	5	20	127
TOLERANCE	1.00-06	1.00-06	1.00-06	1.00-06	1.00e-06
USER CPU TIME (SECS)	6.30-01	2.70-01	6.17+00	9.70-01	3.55e+00
NO. OF TRIPLETS FOUND	13	10	10	10	10
NO. OF TRIPLETS SOUGHT	10	10	10	10	10
NO. MULTIPLICATIONS BY A	64	65	291	325	3088
NO. MULT. BY TRANSPOSE(A)	64	55	273	315	3088
LEFT END OF INTERVAL	-1.00-30	-1.00-30	-	-	-
RIGHT END OF INTERVAL	1.00-30	1.00-30	-	-	-
INITIAL BLOCKSIZE	-	-	4	5	14
FINAL BLOCKSIZE	-	-	1	1	4
MAXIMUM SUBSPACE BOUND	-	-	60	20	-
FINAL SUBSPACE BOUND	-	-	51	11	-
MAX CHEBYSHEV DEGREE	-	-	-	-	14
JOB PARM FOR TMS1(2)	-	-	-	-	-
RESID. REDUCTION TOL.	-	-	-	-	-

Figure 2: `svdin.sum1` file generated by `svdrun` script using output files generated by `svdrun` script.

PROGRAM	sis2	tms1	tms2
DATASETS	'belladit'	'belladit'	'belladit'
FILENAME	belladit.out6	belladit.out7	belladit.out8
DATE	Feb 19 1993	Feb 19 1993	Feb 19 1993
MAX. NO. OF ITERATIONS	80	80	80
ORDER OF EIGENSYSTEM	82	456	82
ROW DIMENSION OF A	374	374	374
COLUMN DIMENSION OF A	82	82	82
AUXILLARY MEMORY(BYTES)	634688	182804	71268
WANT S-VECTORS? [T/F]	T	T	T
NO. OF STEPS/ITERATIONS	35	25	18
TOLERANCE	1.00e-06	7.80+01	7.80+01
USER CPU TIME (SECS)	8.20e-01	6.98+00	1.77+00
NO. OF TRIPLETS FOUND	10	10	10
NO. OF TRIPLETS SOUGHT	10	10	10
NO. MULTIPLICATIONS BY A	992	1179	755
NO. MULT. BY TRANSPOSE(A)	992	1427	745
LEFT END OF INTERVAL	-	-	-
RIGHT END OF INTERVAL	-	-	-
INITIAL BLOCKSIZE	16	12	12
FINAL BLOCKSIZE	6	2	2
MAXIMUM SUBSPACE BOUND	-	-	-
FINAL SUBSPACE BOUND	-	-	-
MAX CHEBYSHEV DEGREE	2	-	0
JOB PARM FOR TMS1(2)	-	1	1
RESID. REDUCTION TOL.	-	1.00+00	1.00+00

Figure 3: `svdin.sum2` file generated by `svdrun` script using output files generated by `svdrun` script.

Data	Application	Columns	Rows	Nonzeros	Density	μ_c	μ_r
APPLE1	IR	44	3206	7722	0.05	175.5	2.4
APPLE2	IR	294	1472	13442	0.03	45.7	9.1
ATGC2	IR	238	3253	54440	0.07	228.7	16.7
CASSERES	IR	117	1453	21597	0.13	184.6	14.9
DULCEC2	IR	94	1299	10285	0.08	109.4	7.9
KASSC2	IR	112	1982	19115	0.09	170.7	9.6
LATERAL2	IR	747	2695	47115	0.02	63.1	17.4
LATERAL5	IR	747	1614	43061	0.04	57.6	26.7
MILLER2	IR	283	3642	60505	0.06	213.8	16.6
VARIETYC1	IR	107	2252	14915	0.06	139.4	6.6
VARIETYC2	IR	107	1567	13545	0.08	126.6	8.6
WMURRAYC1	IR	242	2869	25456	0.04	105.2	8.9
WMURRAYC2	IR	242	1997	23172	0.05	97.9	11.9
APFIRO	LP	28	32	88	0.09	3.1	2.8
BEACONFD	LP	174	262	3476	0.08	19.9	13.3
DEGEN2	LP	445	534	4449	0.02	9.9	8.3
E226	LP	224	282	2767	0.04	12.4	9.8
ETAMACRO	LP	401	688	2489	0.009	6.2	3.6
FFFFF800	LP	525	854	6235	0.01	11.9	7.3
GROW15	LP	301	645	5665	0.03	18.8	8.8
NZFRI	LP	624	3521	15903	0.007	25.4	4.5
PILOT4	LP	411	1000	5145	0.01	12.5	5.1
SCFXM1	LP	331	457	2612	0.02	7.9	5.7
SCTAP1	LP	301	480	2052	0.01	6.8	4.3
SCSD6	LP	148	1350	5666	0.03	38.2	4.2
SEBA	LP	516	1028	4874	0.009	9.4	4.7
SHELL	LP	537	1775	4900	0.005	9.1	2.8
STAIR	LP	357	467	3857	0.02	10.8	8.3
STANDATA	LP	360	1075	3038	0.008	8.4	2.8

Table 12: SVDPACKC Sparse Matrix Test Suite. IR $\equiv$ Information Retrieval, LP $\equiv$ Linear Programming.

5.2 Machine Specifications

Some of the machine specifications for the workstations used in our benchmark experiments are given in Table 5.2. It is advisable to always use the math coprocessor (MC68881) for SVDPACKC on the Macintosh II/fx. Without floating-point hardware, SVDPACKC programs can exhaust as much as 6 times the normal CPU seconds required with a coprocessor.

Model	Macintosh II/fx	Sun-4/490
OS	System 7; MPW 3.3	Sun OS 4.1
Memory	32 Mbytes RAM	32 Mbytes RAM
C Compiler	MPW C	GNU C (gcc)
Compiler Options	-mc68020 -mc68881 -elems881	-O
Linpack MFLOPS (N=100)	0.37	3.6

Table 13: Machine Specifications for SVDPACKC Benchmarks.

5.3 Results

The elapsed user CPU times (in seconds) for SVDPACKC routines executed on the Macintosh II/fx are illustrated in Figures 4 through 8. We also provide tabulated results for both the Macintosh II/fx and Sun-4/490 in Tables 14 through 18 in Appendix A (Section 8) along with the number of approximated singular triplets, p, having residual norms (4) no larger than 10^{-6}. Figures 4 through 7 (and Tables 14 through 17) reflect timings using the IR matrices from Table 12, while Figure 8 (and Table 18) show elapsed user CPU times for sis2 and las2 on the 16 LP matrices from Table 12. The input parameters used for each SVDPACKC routine in our benchmarks are provided in the `svdin.bench` file[5].

As observed in [4] and [5], las2 is by far the fastest sequential method for computing several of the largest singular triplets of large sparse matrices. This, of course, assumes there is no loss of accuracy in approximating eigenpairs of the matrix $A^T A$, which is the case for the matrices comprising our test suite in Table 12. Among competitive Lanczos-based SVDPACKC methods for computing several of the largest singular triplets of the IR matrices, las2 is on average 5 and 9 times faster than las1 and bls2, respectively, on the Macintosh II/fx. On the Sun-4/490, las2 is about 4 and 6 times faster than las1 and bls2, respectively. Among subspace iteration-based methods, we observe sis2 to be on average about 3.5 and 1.5 times faster than sis1 and tms2, respectively on the Macintosh II/fx. On the Sun-4/490, sis2 is on average about 3.5 and 2.25 times faster than sis1 and tms2, respectively. However, las2 is still about 8 and 5 times faster than sis2 across both machines considered.

For the 16 LP matrices, we find (see Figure 8 and Table 18) the most competitive methods from the Lanczos-based group {las1, las2, bls1, bls2} and subspace iteration-based group {sis1, sis2, tms1, tms2} to be las2 and sis2, respectively. On both the Macintosh II/fx and Sun-4/490, las2 is on average 5 times faster than sis2 when computing as many as 50 of the largest singular triplets for the LP matrices arising from linear programming applications.

From Figures 4 and 8 (and Tables 14 and 18), we also observe that las2 on the Macintosh II/fx averages from 3.75 to 7.8 times slower than las2 on the Sun-4/490. This reflects a significant cost-performance benefit given the affordability and availability of Macintosh computers.

[5]ASCII file in SVDPACKC distribution package.

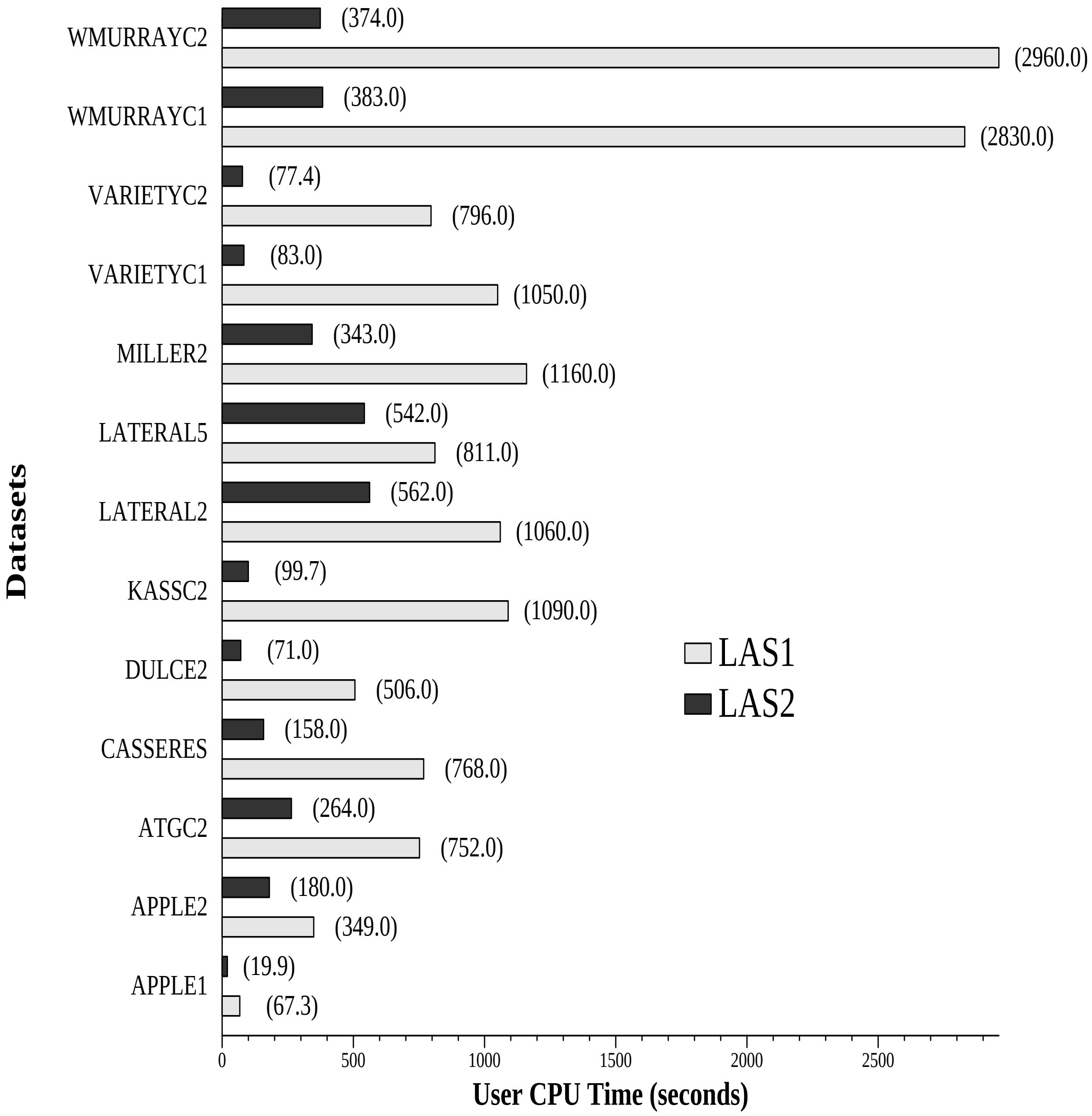

Figure 4: User CPU time (in seconds) expired by the single-vector Lanczos methods (las1, las2) on the Macintosh II/fx when computing singular triplets of the IR matrices.

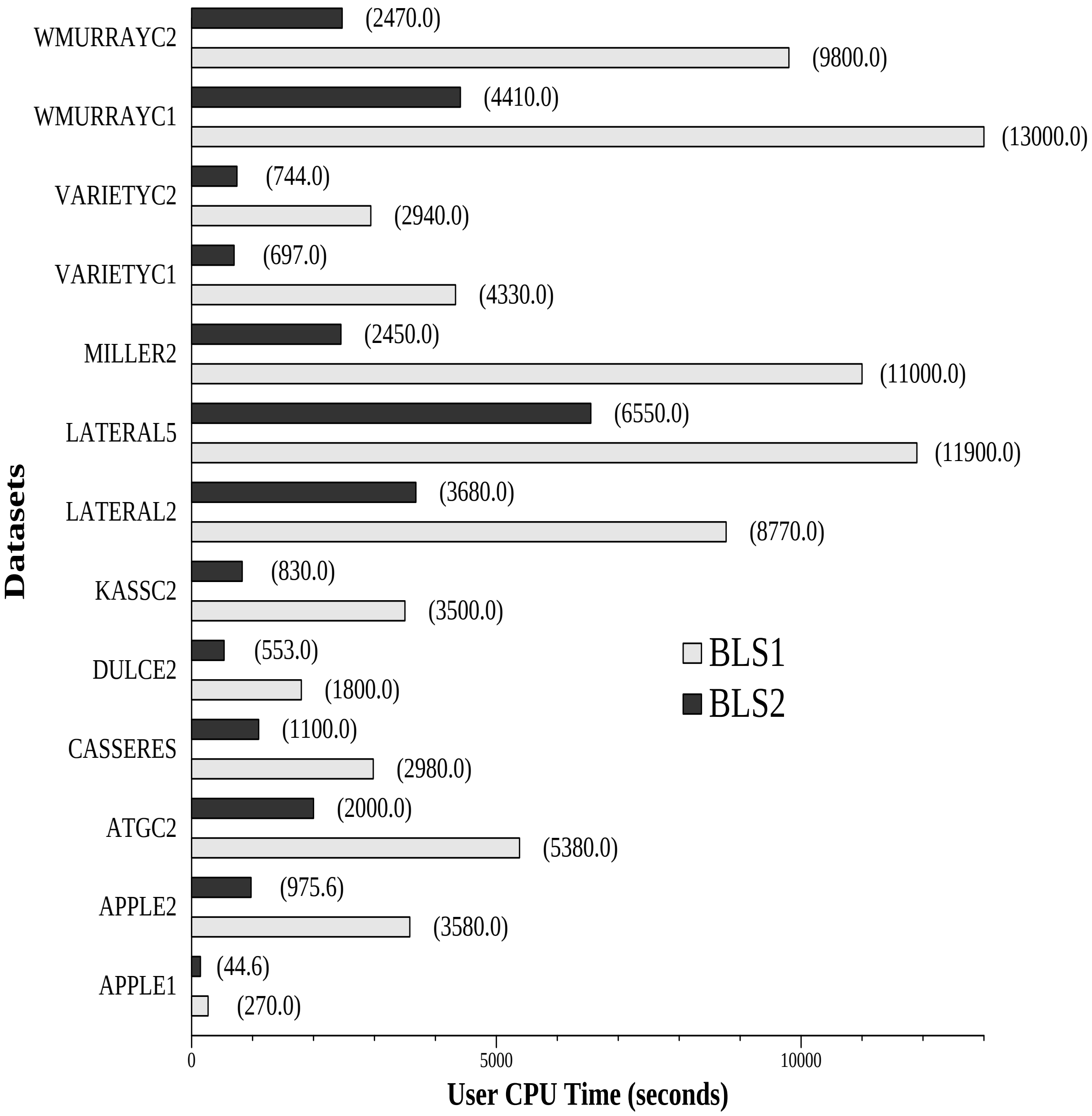

Figure 5: User CPU time (in seconds) expired by the block Lanczos methods (bls1, bls2) on the Macintosh II/fx when computing singular triplets of the IR matrices.

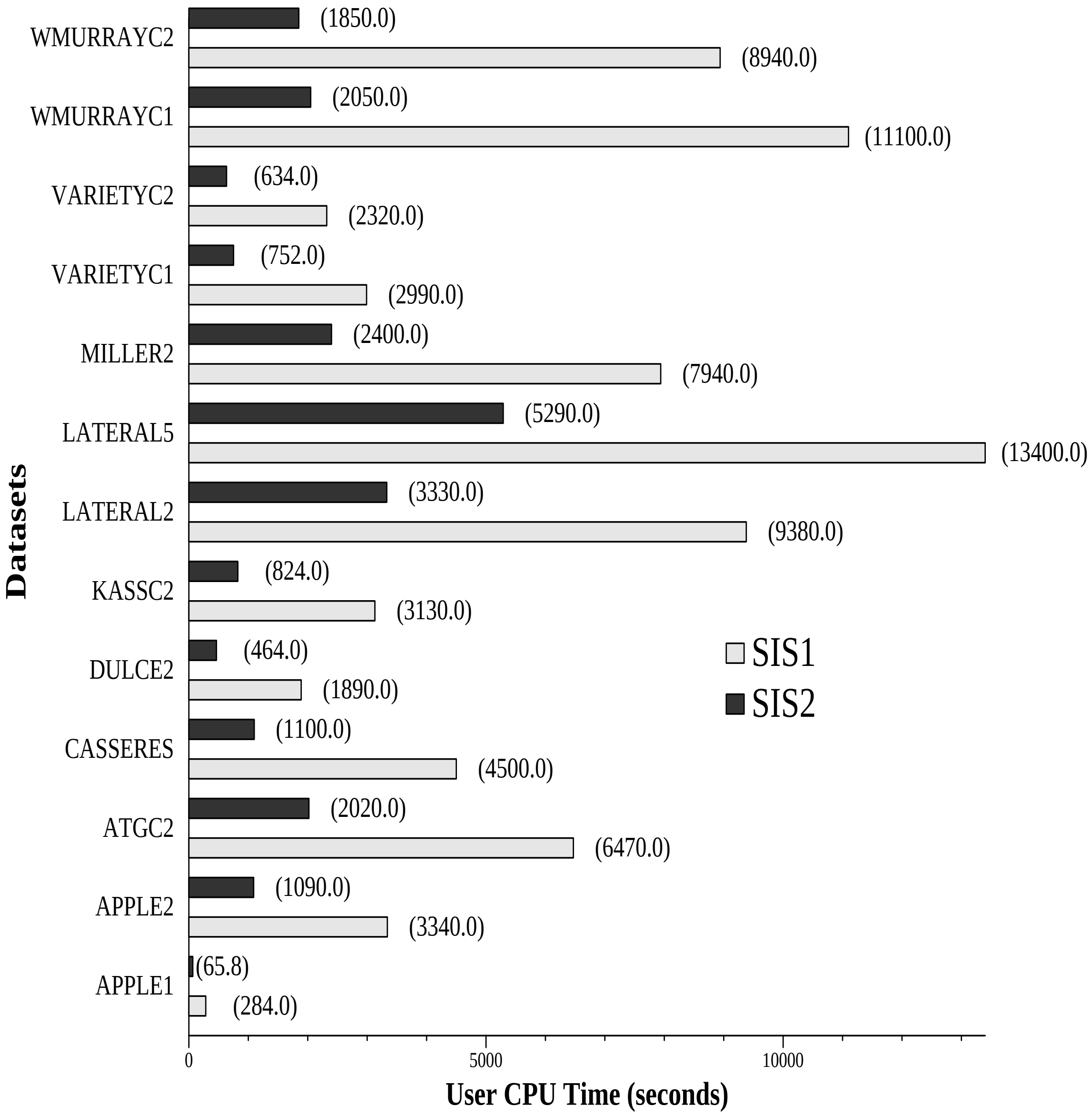

Figure 6: User CPU time (in seconds) expired by the subspace iteration methods (sis1, sis2) on the Macintosh II/fx when computing singular triplets of the IR matrices.

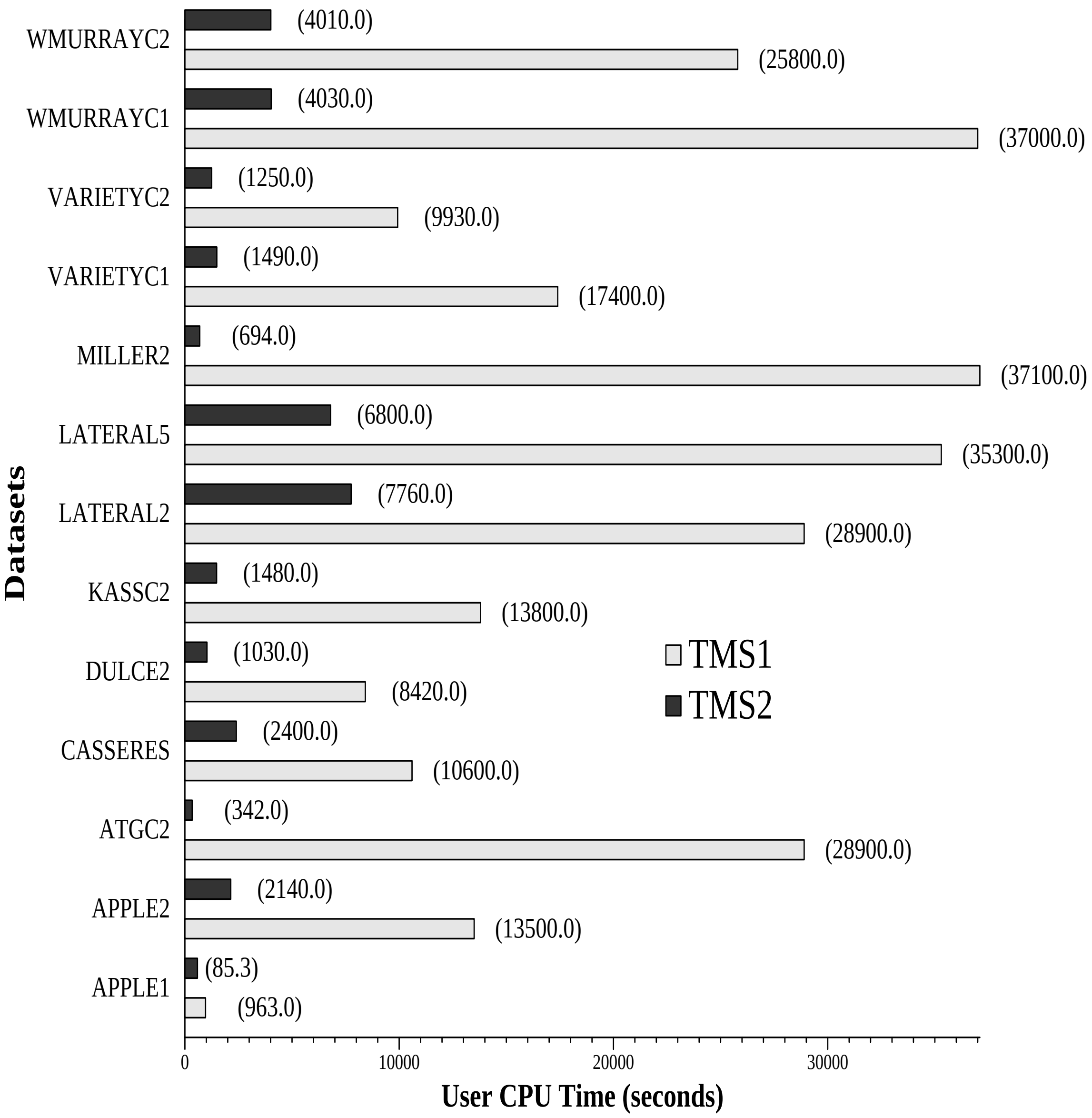

Figure 7: User CPU time (in seconds) expired by the trace minimization methods (tms1, tms2) on the Macintosh II/fx when computing singular triplets of the IR matrices.

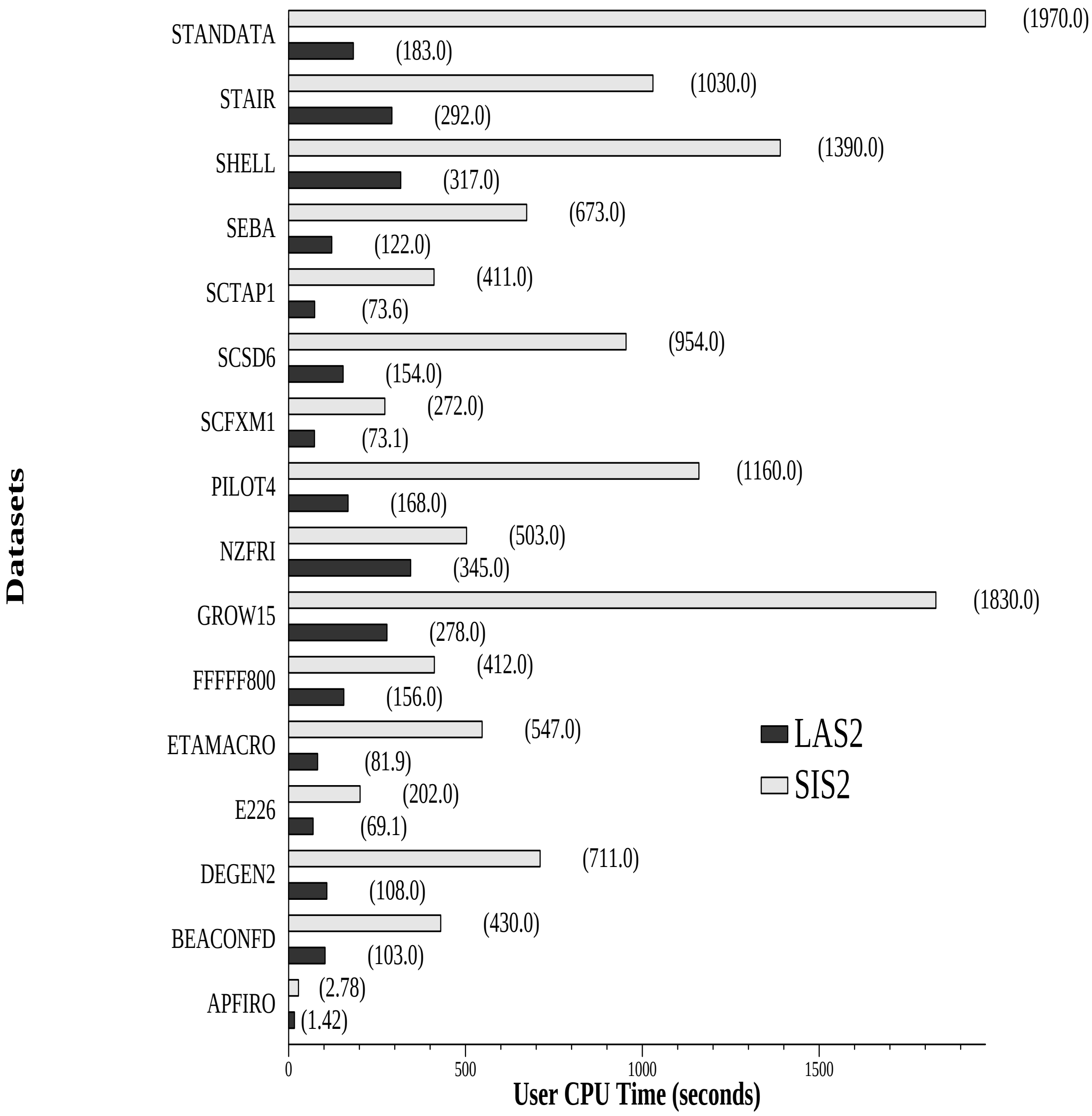

Figure 8: User CPU time (in seconds) expired by sis2 and las2 on the Macintosh II/fx when computing singular triplets of the LP matrices.

6 Future Work

For determining the singular value decomposition of extremely large sparse matrices using a modest computing environment, we anticipate the development of an out-of-core distributed version of SVDPACKC which ports to a heterogeneous network of workstations. Implementations on massively-parallel computer systems such as the MasPar MP-2 and Thinking Machines CM-5 are planned as well. Future algorithmic concerns include the use of alternative re-orthogonalization strategies for bls1 and bls2, and the development of techniques for computing large rank updates to the singular value decomposition of unstructured sparse matrices.

The original SVDPACK Fortran-77 source code and documentation may be obtained through the NETLIB facility maintained by the University of Tennessee and Oak Ridge National Laboratory. Users should send the electronic mail message *send index from svdpack* to `netlib@ornl.gov` to get a listing of the methods and associated files composing the library. The SVDPACKC library should be available in NETLIB by Summer 1993.

7 Acknowledgements

The authors would like to thank Dulce Ponceleon at Apple Computer Inc., Cupertino, CA, for her collaborative efforts in constructing the test suite of sparse matrices from information retrieval applications.

References

[1] Bauer, F. L. Das Verfahren der Treppeniteration und verwandte Verfahren zur Lösung algebraischer Eigenwertprobleme. *ZAMP 8* (1957), 214–235.

[2] Berry, M., and Sameh, A. An overview of parallel algorithms for the singular value and dense symmetric eigenvalue problems. *Journal of Computational and Applied Mathematics 27* (1989), 191–213.

[3] Berry, M. W. *Multiprocessor Sparse SVD Algorithms and Applications.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[4] Berry, M. W. Large scale singular value computations. *International Journal of Supercomputer Applications 6*, 1 (1992), 13–49.

[5] Berry, M. W. SVDPACK: A Fortran-77 software library for the sparse singular value decomposition. Tech. Rep. CS-92-159, University of Tennessee, Knoxville, TN, June 1992.

[6] Cullum, J. K., and Donath, W. E. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and corresponding eigenspace of large sparse real symmetric matrices. In *Proceedings of 1974 IEEE Conf. on Decision and Control* (1974), pp. 505–509.

[7] Cullum, J. K., and Willoughby, R. A. *Lanczos Algorithm for Large Symmetric Eigenvalue Computations, Volume 1 Theory.* Birkhäuser, Boston, 1985.

[8] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 6 (1990), 391–407.

[9] Dongarra, J., Croz, J. D., Hammarling, S., and Hanson, R. An extended set of fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software 14*, 1 (1988), 1–17.

[10] Dongarra, J., and Sorensen, D. A fast algorithm for the symmetric eigenvalue problem. *SIAM Journal of Statistical and Scientific Computing 8*, 2 (1987), s139–s154.

[11] Duff, I. S., Grimes, R. G., and Lewis, J. G. Sparse matrix test problems. *ACM Trans. Math. Software 15* (1989), 1–14.

[12] Dumais, S., Furnas, G., and Landauer, T. Using latent semantic analysis to improve access to textual information. In *Proceedings of Computer Human Interaction '88* (1988).

[13] Dumais, S. T. Improving the retrieval of information form external sources. *Behavior Research Methods, Instruments, & Computers 23*, 2 (1991), 229–236.

[14] Gallivan, K., Jalby, J., and Meier, U. The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory. *SIAM J. Sci. Stat. 18*, 6 (1987), 1079–1084.

[15] Golub, G., and Kahan, W. Calculating the singular values and pseudoinverse of a matrix. *SIAM Journal of Numerical Analysis 2*, 3 (1965), 205–224.

[16] Golub, G., and Loan, C. V. *Matrix Computations*, second ed. Johns-Hopkins, Baltimore, 1989.

[17] Golub, G., Luk, F., and Overton, M. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Transactions on Mathematical Software 7*, 2 (1981), 149–169.

[18] Golub, G., and Reinsch, C. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation II, Linear Algebra*. Springer-Verlag, New York, 1971.

[19] Golub, G. H., and Underwood, R. R. The block Lanczos method for computing eigenvalues. In *Mathematical Software III*. Academic Press, New York, 1977, pp. 361–377.

[20] Hestenes, M. R. Inversion of matrices by biorthogonalization and related results. *Journal of the Society for Industrial and Applied Mathematics 6* (1958), 51–90.

[21] Kaniel, S. Estimates for some computational techniques in linear algebra. *Math. Comp. 20* (1966), 369–378.

[22] Luenberger, D. G. *Introduction to Linear and Nonliner Programming*. Addison-Wesley, Reading, MA, 1973.

[23] Lustig, I. J. An analysis of an available set of linear programming test problems. Tech. Rep. SOL 87-11, Department of Operations Research, Stanford University, Stanford, CA, August 1987.

[24] Mirsky, L. Symmetric gage functions and unitarilly invariant norms. *Quarterly Journal of Mathematics 11* (1960), 50–59.

[25] Paige, C. C. Error analysis of the Lanczos algorithms for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl. 18* (1976), 341–349.

[26] Parlett, B. *The Symmetric Eigenvalue Problem.* Prentice Hall, Englewood Cliffs, NJ, 1980.

[27] Parlett, B., and Scott, D. The Lanczos algorithm with selective reorthogonalization. *Math. Comp. 33* (1979), 217–238.

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

[29] Saad, Y. On the rates of convergence of the Lanczos and the block-Lanczos methods. *SIAM J. Numer. Anal. 17* (1980), 687–706.

[30] Sameh, A. H., and Wisniewski, J. A. A trace minimization algorithm for the generalized eigenvalue problem. *SIAM Journal of Numerical Analysis 19*, 6 (1982), 1243–1259.

[31] Simon, H. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Lin. Alg. and Its. Appl. 61* (1984), 101–131.

[32] Smith, B., et al. *Matrix Eigensystem Routines - EISPACK Guide*, second ed. Spinger-Verlag, Berlin, 1976.

[33] Underwood, R. R. *An iterative block Lanczos method for the solution of large sparse symmetric eigenproblem.* PhD thesis, Stanford University, Stanford, CA, 1975.

[34] Varga, R. S. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1962.

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

[36] Wilkinson, J. H. Inverse iteration in theory and in practice. In *Symposia Mathematica X.* Academic Press, London, 1972.

[37] Wisniewski, J. A. *On solving the large sparse generalized eigenvalue problem.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1981.

8 Appendix A: SVDPACKC Benchmarks

		las1		las2	
Matrix	p	Sun-4/490	Mac. II/fx	Sun-4/490	Mac. II/fx
APPLE1	10	64.1	67.3	4.1	19.9
APPLE2	50	275.0	349.0	45.3	180.0
ATGC2	50	136.0	752.0	61.2	264.0
CASSERES	50	134.0	768.0	35.0	158.0
DULCEC2	50	86.5	506.0	14.0	71.0
KASSC2	50	180.0	1090.0	21.8	99.7
LATERAL2	50	154.0	1060.0	73.8	562.0
LATERAL5	75	500.0	811.0	224.0	542.0
MILLER2	50	215.0	1160.0	89.5	343.0
VARIETYC1	50	175.0	1050.0	18.0	83.0
VARIETYC2	50	139.0	796.0	16.6	77.4
WMURRAYC1	75	645.0	2830.0	78.6	383.0
WMURRAYC2	75	524.0	2960.0	71.8	374.0
Total Time (sec)		3227.6	14191.3	753.7	3157.0
Average Time (sec)		248.3	1091.6	57.9	242.8

Table 14: User CPU time (in seconds) expired by the single-vector Lanczos methods (las1, las2) on the Sun-4/490 and the Macintosh II/fx when computing the p-largest singular triplets for the IR matrices.

		bls1		bls2	
Matrix	p	Sun-4/490	Mac. II/fx	Sun-4/490	Mac. II/fx
APPLE1	10	47.5	270.0	9.78	44.6
APPLE2	50	588.0	3580.0	244.0	975.0
ATGC2	50	840.0	5380.0	333.0	2000.0
CASSERES	50	599.0	2980.0	247.0	1100.0
DULCEC2	50	268.0	1800.0	107.0	533.0
KASSC2	50	674.0	3500.0	209.0	830.0
LATERAL2	50	1320.0	8770.0	598.0	3680.0
LATERAL5	75	1780.0	11900.0	1080.0	6550.0
MILLER2	50	1660.0	11000.0	405.0	2450.0
VARIETYC1	50	671.0	4330.0	183.0	697.0
VARIETYC2	50	532.0	2940.0	163.0	744.0
WMURRAYC1	75	2670.0	13000.0	718.0	4410.0
WMURRAYC2	75	1500.0	9800.0	538.0	2470.0
Total Time (sec)		13149.5	79250.0	4834.7	26483.6
Average Time (sec)		1011.5	6096.1	371.9	2037.2

Table 15: User CPU time (in seconds) expired by the block Lanczos methods (bls1, bls2) on the Sun-4/490 and the Macintosh II/fx when computing the p-largest singular triplets for the IR matrices.

Matrix	p	sis1		sis2	
		Sun-4/490	Mac. II/fx	Sun-4/490	Mac. II/fx
APPLE1	10	47.1	284.0	11.4	65.8
APPLE2	50	556.0	3340.0	177.0	1090.0
ATGC2	50	1160.0	6470.0	380.0	2020.0
CASSERES	50	781.0	4500.0	194.0	1100.0
DULCEC2	50	300.0	1890.0	74.4	464.0
KASSC2	50	529.0	3130.0	143.0	824.0
LATERAL2	50	1650.0	9380.0	579.0	3330.0
LATERAL5	75	2300.0	13400.0	876.0	5290.0
MILLER2	50	1390.0	7940.0	422.0	2400.0
VARIETYC1	50	487.0	2990.0	124.0	752.0
VARIETYC2	50	374.0	2320.0	103.0	634.0
WMURRAYC1	75	1890.0	11100.0	352.0	2050.0
WMURRAYC2	75	1520.0	8940.0	315.0	1850.0
Total Time (sec)		12984.1	75684.0	3750.8	21869.8
Average Time (sec)		998.7	5821.8	288.5	1682.3

Table 16: User CPU time (in seconds) expired by the subspace iteration methods (sis1, sis2) on the Sun-4/490 and the Macintosh II/fx when computing the p-largest singular triplets for the IR matrices.

Matrix	p	tms1 Sun-4/490	tms1 Mac. II/fx	tms2 Sun-4/490	tms2 Mac. II/fx
APPLE1	10	139.0	963.0	15.0	85.3
APPLE2	50	1840.0	13500.0	320.0	2140.0
ATGC2	50	4160.0	28900.0	1090.0	342.0
CASSERES	50	1470.0	10600.0	401.0	2400.0
DULCEC2	50	1110.0	8420.0	158.0	1030.0
KASSC2	50	1850.0	13800.0	253.0	1480.0
LATERAL2	50	4180.0	28900.0	1190.0	7760.0
LATERAL5	75	4740.0	35300.0	1830.0	6800.0
MILLER2	50	5430.0	37100.0	958.0	964.0
VARIETYC1	50	2350.0	17400.0	242.0	1490.0
VARIETYC2	50	134.0	9930.0	194.0	1250.0
WMURRAYC1	75	7220.0	37000.0	655.0	4030.0
WMURRAYC2	75	3520.0	25800.0	1120.0	4010.0
Total Time (sec)		38143.0	267613.0	8426.0	33781.3
Average Time (sec)		2934.0	20585.6	648.1	2598.6

Table 17: User CPU time (in seconds) expired by the trace minimization methods (tms1, tms2) on the Sun-4/490 and the Macintosh II/fx when computing the p-largest singular triplets for the IR matrices.

Matrix	p	sis2		las2	
		Sun-4/490	Mac. II/fx	Sun-4/490	Mac. II/fx
APFIRO	10	.370	2.78	.190	1.42
BEACONFD	50	55.3	430.0	13.6	103.0
DEGEN2	50	93.9	711.0	13.9	108.0
E226	50	26.1	202.0	9.40	69.1
ETAMACRO	50	69.1	547.0	11.4	81.9
FFFFF800	50	55.4	412.0	21.4	156.0
GROW15	50	240.0	1830.0	35.3	278.0
NZFRI	50	47.7	503.0	38.3	345.0
PILOT4	50	148.0	1160.0	22.2	168.0
SCFXM1	50	33.8	272.0	9.20	73.1
SCSD6	50	129.0	954.0	20.8	154.0
SCTAP1	50	50.4	411.0	9.14	73.6
SEBA	50	86.6	673.0	16.0	122.0
SHELL	50	179.0	1390.0	41.8	317.0
STAIR	50	137.0	1030.0	35.0	292.0
STANDATA	50	253.0	1970.0	26.0	183.0
Total Time (sec)		1604.7	12497.8	323.6	2525.1
Average Time (sec)		100.3	781.1	20.2	157.8

Table 18: User CPU time (in seconds) expired by sis2 and las2 on the Sun-4/490 and the Macintosh II/fx when computing the p-largest singular triplets for the LP matrices.

9 Appendix B: Sparse Matrix Storage Formats

All matrices in the original Harwell-Boeing sparse matrix test collection are stored in a column oriented compact format (Fortran-77 influence) where only the entries corresponding to nonzero values are stored. The row indices and corresponding nonzero numerical values are stored by columns with a column start vector pointing to the beginning of each column. Symmetric, skew symmetric, and Hermitian matrices have only the entries of the lower triangle (including the diagonal) stored. Right-hand-side vectors for linear systems are stored in a full arrays (not necessary for SVDPACKC use).

Each matrix is written in a standard format with a 4 line header record followed by up to 4 logical records containing, in order, the column start pointers, the row indices, the numerical values, and the right-hand-side matrix. The records containing the numerical values and right-hand-side matrix are optional. The right-hand-side matrix can only be present when the numerical values are present. All lines are restricted to 80 columns. For SVDPACKC routines, you need only supply the column start pointers, row indices, and numerical values of all nonzeros.

The header record consists of 4 lines of data. The first line contains the 72 character title and a 8 character key by ehich the matrices are referenced. The second line contains the number of lines for each of the following 4 records as well as the total number of lines, excluding the header record, for the matrix. The third line contains a 3 character string denoting the matrix type as well as the number of rows, columns, nonzeroes, and right-hand-sides vectors for the matrix. The fourth line contains the 4 variable formats for the following 4 logical records. The exact formats are

```
Line 1 ( A72, A8 )
Col.  1 - 72 Title
Col. 73 - 80 Key

Line 2 ( 5I14 )
Col.  1 - 14 Total No. of lines excluding header
Col. 15 - 28 No. of lines for pointers
Col. 29 - 42 No. of lines for row indices
Col. 43 - 56 No. of lines for numerical values
```

```
Col. 57 - 70 No. of lines right-hand-sides

Line 3 ( A3, 11x, 4I14 )
Col.  1 -  3 Matrix type
Col. 15 - 28 No. of rows
Col. 29 - 42 No. of columns
Col. 43 - 56 No. of nonzeroes
Col. 57 - 70 No. of right-hand-sides

Line 4 ( 2A16, 2A20 )
Col.  1 - 16 Format for pointers
Col. 17 - 32 Format for row indices
Col. 33 - 52 Format for numerical values
Col. 53 - 72 Format for right-hand-sides

The 3 character type field describes the matrix type.  The
following table lists the allowed values for each of the 3
characters.  As an example of the type field 'rsa' denotes that
the matrix is real, symmetric and assembled.

First Character: r Real Matrix
c Complex Matrix
p Pattern Only (no values supplied)

Second Character: s Symmetric
u Unsymmetric
h Hermitian
z Skew symmetric
r Rectangular

Third Character: a Assembled
f Unassembled Finite Elements
```

For SVDPACKC, several of the fields specified above are not necessary for input via the `fscanf()` C function. For example, in las2 the required header information is read via following lines

```
fscanf (fp_in2, "%72c%*s%*s%*s%ld%ld%ld%*d",
        title, &nrow, &ncol, &nnzero);
fscanf (fp_in2, "%*s %*s %*s %*s");
```

An appropriate header for any of the SVDPACKC codes might look like

```
Bellcore ADI Linguistics Data                    belladit
#
rra       374            82           1343         0
          (10i8)         (10i8)       (8f10.3)     (8f10.3)
```

Notice that the information in the second header line of the original Fortran Harwell-Boeing format (line counts) is not needed by SVDPACKC and we simply replace that line by a single character (such as `#`). Although the formats listed in the fourth header line are also not required by SVDPACKC, their presence specifies, for example, the number of decimal digits contained in the numerical values.

Before providing an example of the Compressed Column Storage (CCS) format used in the Harwell-Boeing format, we review the 3 arrays used to define an arbitrary sparse matrix. The CCS format is specified by the 3 arrays $\{\texttt{value}, \texttt{rowind}, \texttt{pointr}\}$, where `rowind` stores the row indices of each nonzero, and `pointr` stores the index of the elements in `value` which start a column of of the matrix A, as they are traversed in a column-wise fashion. The `rowind` vector stores the row indexes of the elements in the `value` array. That is, if $\texttt{val(k)} = A_{ij}$ then $\texttt{rowind[k]} = i$, for $i = 1, 2, \ldots, m$, and $j = 1, 2, \ldots, n$ for m rows and n columns. The `pointr` array stores the locations in the `value` array that start a column, that is, if $\texttt{value[k]} = A_{ij}$ then $\texttt{pointer[i]} \leq k < \texttt{pointr[i + 1]}$. By convention, we define $\texttt{pointr[n + 1]} = nnz + 1$, where nnz is the number of nonzeros in the $m \times n$ matrix A. The storage savings for this approach is significant. Instead of storing $m \times n$ elements, we need only $2 \times nnz + n + 1$ storage locations. To be compatible with the original Fortran-based sparse format, SVDPACKC routines assume that the integer values read into the `pointr` and `rowind` arrays satisfy $1 \leq \texttt{pointr},\ \texttt{rowind} \leq nnz + 1$ rather than $0 \leq \texttt{pointr},\ \texttt{rowind} \leq nnz$.

The following lines from las2 demonstrate how the CCS arrays are read via `fcanf()`. The elements of arrays `rowind` and `point` are decremented by 1 so that both arrays having starting index 0 within las2.

```
for (i = 0; i <= ncol; i++) fscanf(fp_in2, "%ld", &pointr[i]);
for (i = 0; i <  ncol; i++) pointr[i] -= 1;

/* define last element of pointr in case it is not */
pointr[i] = nnzero;

for (i = 0; i < nnzero; i++) fscanf(fp_in2, "%ld", &rowind[i]);
for (i = 0; i < nnzero; i++) rowind[i] -= 1;
for (i = 0; i < nnzero; i++) fscanf(fp_in2, "%lf", &value[i]);
```

Consider the following 6×6 matrix A, where

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}.$$

The CCS format for the matrix A above is given by

`value`	10	3	3	9	7	8	4	8	$8 \cdots 9$	2	3	13	-1
`rowind`	1	2	4	2	3	5	6	3	$4 \cdots 5$	6	2	5	6

`pointr`	1	4	8	10	13	17	20

,

and a corresponding input file for SVDPACKC is given below. Note that SVDPACKC does not require fields in the header to match those in the specifications of the original Harwell-Boeing (Fortran) header. Alternative sparse matrix formats can be used, of course, with SVDPACKC. An analogous Compressed Row Storage (CRS) format based on row-wise traversal is another possibility.

```
Sample Input Matrix for SVDPACKC          sample
#
rra       6       6         18          0
          (8i6)   (8i6)     (8f6.2)     (8f6.2)
     1     4     8    10    13    17    20
     1     2     4     2     3     5     6     3
     4     5     1     4     5     6     2     5
     6
 10.00  3.00  3.00  9.00  7.00  8.00  4.00  8.00
  8.00  7.00  7.00  9.00 -2.00  5.00  9.00  2.00
  3.00 13.00 -1.00
```

10 Appendix C: Binary Output Files

The binary output files generated by SVDPACKC primarily contain the approximate singular values and corresponding singular vectors. In the table below, we list the contents of each binary output file of the form `MMvN`, where `N` = 1,2, and `MM` defines the method used (see Table 2 in Section 3.2). Let u_i, σ_i, v_i denote the i-th largest singular triplet for an $m \times n$ sparse matrix so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, and let k denote the number of singular triplets written to file.

Filename	Contents (in order)
`blv1`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$
`blv2`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$
`lav1`	$m + n$
	js (No. of Lanczos steps)
	$kappa$ (Residual Tolerance)
	$v_k, v_{k-1}, \ldots, v_2, v_1$
	$u_k, u_{k-1}, \ldots, u_2, u_1$
`lav2`	n
	js (No. of Lanczos steps)
	$kappa$ (Residual Tolerance)
	$v_k, v_{k-1}, \ldots, v_2, v_1$
	$u_k, u_{k-1}, \ldots, u_2, u_1$
`siv1`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$
`siv2`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$
`tmv1`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$
`tmv2`	$u_1, v_1, u_2, v_2, \ldots, u_k, v_k$