ORNL/CSD-48

UCC-ND

NUCLEAR DIVISION

UNION CARBIDE

Block Lanczos Software for Symmetric Eigenvalue Problems

D. S. Scott

OPERATED BY
UNION CARBIDE CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A05 Microfiche A01

ORNL/CSD-48
Distribution Category UC-32

Contract No. W-7405-eng-26

COMPUTER SCIENCES DIVISION

BLOCK LANCZOS SOFTWARE FOR SYMMETRIC EIGENVALUE PROBLEMS

D. S. Scott

Sponsor: D. A. Gardiner
Originator: D. S. Scott

Date Published: November 1979

UNION CARBIDE CORPORATION, NUCLEAR DIVISION
operating the
Oak Ridge Gaseous Diffusion Plant . Oak Ridge National Laboratory
Oak Ridge Y-12 Plant . Paducah Gaseous Diffusion Plant
for the
DEPARTMENT OF ENERGY

Table of Contents

Table of Contents (Cont'd)

Table of Contents (Cont'd)

ABSTRACT

The Lanczos algorithm is a powerful method of computing a few eigenvalues and eigenvectors of a large sparse symmetric matrix. Selective orthogonalizing is an efficient method of maintaining the stability of the algorithm. This report documents the design and implementation of two distinct block Lanczos algorithms with selective orthogonalization.

INTRODUCTION

This report gives detailed documentation of the implementation of a package of subroutines for solving large sparse symmetric eigenvalue problems. There are two driver subroutines. The user guides for these two subroutines are given in Appendix 1 and it is recommended that the reader become familiar with the user guides before reading the body of the report.

The package is available in both single and double precision. This report explicitly documents the single precision version although periodic references to the double precision version are also made. The four major changes between single and double precision are:

1. The names of the subroutines are different.
2. All real variables and constants are double precision (except URAND, see 4).
3. There are discrepancies in the sequence numbers due to the extra continuation lines needed in the double precision version.
4. Modifications are necessary due to the fact that the function URAND exists only in single precision.

1. THE LANCZOS ALGORITHM

This section gives a brief synopsis of the Lanczos algorithm. No proofs are given and only those results which are pertinent to the rest of the report are included. The reader is directed to Scott [12] for a more thorough treatment of the subject.

The Lanczos algorithm is an iterative procedure for computing eigenvalues of a symmetric matrix A, which starts with an arbitrary vector r_0 and $q_0 \equiv 0$. Then for j = 1,2,...DO 1 to 5,

1. $\beta_j = \|r_{j-1}\|$
2. If $\beta_j = 0$ stop
   else $q_j = r_{j-1}/\beta_j$
3. $u_j = Aq_j - q_{j-1}\beta_j$
4. $\alpha_j = q_j * u_j$
5. $r_j = u_j - q_j\alpha_j$.

Note that the only way that the matrix A enters the calculation is through the formation of the product Aq_j. This is a very attractive feature of the algorithm with respect to sparse matrices since A may be stored in any compact manner which permits the formation of matrix vector products. Note also that only the previous two Lanczos vectors (q's) are needed at any step j.

There are several mathematically equivalent formulations of the Lanczos algorithm. The one given here is the most stable numerically as shown in Paige [8].

Defining $Q_j = (q_1, q_2, \ldots q_j)$ and

$$T_j = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \cdot & \cdot & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \beta_j \\ & & & & \beta_j & \alpha_j \end{bmatrix}$$

then it can be shown that, in exact arithemtic,

$$AQ_j - Q_j T_j = r_j e_j^*$$

and

$$1 - Q_j^* Q_j = 0,$$

where $e_j^* = (0,0,\ldots,1)$ is a j-vector and $r_j e_j^*$ is a compact way of writing an n x j matrix all of whose entries are zero except the last column which is r_j.

The columns of Q_j are an orthonormal basis for the Krylov subspace,

$$K_j(q_1) = \text{span}(q_1, Aq_1, \ldots, A^{j-1} q_1) \ .$$

Multiplying (1) by Q_j^*, we find that since $Q_j^* r_j = 0$,

$$T_j = Q_j^* A Q_j \ .$$

Let $T_j = S_j \Theta_j S_j^*$, with $S_j^* = S_j^{-1}$ and $\Theta_j = \mathrm{diag}(\theta_1^{(j)}, \theta_2^{(j)}, \ldots, \theta_j^{(j)})$, be the spectral decomposition of T_j. Let $Y_j = (y_1^{(j)}, y_2^{(j)}, \ldots, y_j^{(j)}) = Q_j S_j$. Then the pairs $(y_i^{(j)}, \theta_i^{(j)})$ are the (optimal) approximations to eigenpairs of A obtained from the Krylov subspace $K_j(q_1)$ by the Rayleigh-Ritz procedure.

Futhermore the residual norm of a Ritz pair $(y_i^{(j)}, \theta_i^{(j)})$ can be computed as

$$\| A y_i^{(j)} - y_i^{(j)} \theta_i^{(j)} \| = \beta_{j+1} |s_{ji}| \equiv \beta_{ji} \qquad (1)$$

where s_{ji} is the $(j,i)^{\underline{\text{th}}}$ element of S_j. Thus the accuracy of the Ritz value $\theta_i^{(j)}$ can be estimated without computing the Ritz vector at all. Only a j x j tridiagonal eigenproblem need be solved and such problems can be solved quickly and accurately using established techniques.

Therefore the Lanczos algorithm can be terminated as soon as the desired Ritz values are sufficiently accurate and the Ritz vectors need only be accumulated at the end. Since only the previous two Lanczos vectors (q's) are needed by the algorithm itself, it is possible to place the earlier Lanczos vectors in secondary store until they are needed for forming the Ritz vectors. Indeed if eigenvectors are not of interest, there is no need to save the Lanczos vectors at all!

We now let the symbols Q, T, Y, etc. stand for the corresponding quantities actually computed on a machine with a relative presision ε.

Unfortunately the simple Lanczos algorithm is unstable when implemented in finite precision. This instability is manifested in the Lanczos vectors by a loss of orthogonality. Indeed the columns of Q become dependent to working precision. It is still possible to compute the eigensystem of T_j and to form the matrix $Y_j = Q_j S_j$ and we will continue to call these quantities Ritz values and Ritz vectors even though they are not the true Ritz pairs derivable from the subspace spanned by the columns of Q_j.

It can be shown (Paige [7] or Scott [12]) that the loss of orthogonality is quite correlated.

> Theorem (Paige) At any step j of the Lanczos algorithm and all i,
>
> $$y_i^{(j)*} q_{j+1} = \gamma_{ji} \varepsilon \|A\| / \beta_{ji}$$
>
> where $\gamma_{ji} \simeq 1$ and β_{ji} is defined as in equation (1).

Thus the quantity β_{ji}, which is the residual norm of the Ritz pair $(y_i^{(j)}, \theta_i^{(j)})$, is important both in determining the accuracy of $\theta_i^{(j)}$ and in measuring the loss of orthogonality at step j. In words

> loss of orthogonality $\leftrightarrow$ convergence .

This theorem is the basis for selective orthogonalization, a scheme for stabilizing the Lanczos algorithm, described in Parlett and Scott [9] and analyzed in detail in Scott [12]. Selective orthogonalization explicitly

orthogonalizes q_{j+1} against any Ritz vector which satisfies

$$\beta_{ji} \leq \sqrt{\epsilon}\|A\| \quad .$$

Such Ritz vectors are called good. If the tolerance is chosen much less than $\sqrt{\epsilon}\|A\|$ then the γ's in Paiges Theorem may grow and β_{ji} is no longer an accurate measure of the loss of orthogonality. See Scott [12] for more details. Since β_{ji} is the residual norm of the Ritz pair $(y_i^{(j)}, \theta_i^{(j)})$, a good Ritz vector is quite accurate, having converged to about half the precision of the machine. Therefore few (if any) of the Ritz pairs will be good at any one step which explains the computational success of the scheme.

The simple Lanczos algorithm (even with selective orthogonalization) suffers from two defects. If any eigenvector of A is orthogonal to q_1 then it will be orthogonal to all subsequent q's and will not be detected by the algorithm. In particular multiple or closely cluster eigenvalues inevitably cause difficulties. Furthermore if the matrix is in secondary storage and is brought into core a slice at a time to form the matrix vector product, it is inefficient to only multiply one vector at each access. For these reasons block Lanczos algorithms are appealing and have been analyzed by Cullum and Donath [1] and Underwood [14].

Block Lanczos replaces each q-vector in the simple algorithm by an orthonormal block of m vectors which we label P_j. m is called the block size. The block algorithm starts with R_0, an arbitrary n x m matrix, and $P_0 \equiv 0$. Then for j = 1,2,...

1. Factor $R_{j-1} = P_j B_j$ where P_j is orthonormal and B_j is upper triangular.
2. If B_j is singular, stop.

3. $U_j = AP_j - P_{j-1}B_j^*$.

4. $A_j = P_j^* U_j$.

5. $R_j = U_j - P_j A_j$.

The matrix $Q_j = (P_1, P_2, \ldots, P_j)$ is orthonormal (in theory) and the matrix

$$T_j = \begin{bmatrix} A_1 & B_2^* & & & & \\ B_2 & A_2 & B_3^* & & & \\ & B_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & B_j^* \\ & & & & B_j & A_j \end{bmatrix}$$

is <u>block</u> tridiagonal. If $T_j = S_j \theta_j S_j^*$ and $Y_j = Q_j S_j$ then the pairs $(y_i^{(j)}, \theta_i^{(j)})$ are the Ritz pairs and

$$\|Ay_i^{(j)} - y_i^{(j)}\theta_i^{(j)}\| = \|B_{j+1} s_{ji}\| ,$$

where s_{ji} is the m-vector of the last m elements of $s_i^{(j)}$, the eigenvector of T_j associated with $\theta_i^{(j)}$. Therefore it is still possible to compute the residual norm of y_i without computing y_i.

Block Lanczos solves the two main problems associated with simple Lanczos. Eigenvalues are missed only when the corresponding eigenvectors

are orthogonal to the subspace spanned by the block of starting vectors. Thus multiple eigenvalues can be found up to m, the block size. Also m vectors are multiplied for each access of the matrix, which is more efficient than simple Lanczos, particularly if the matrix is in secondary storage.

However, block Lanczos suffers from the same loss of orthogonality as simple Lanczos. The generalization of Paige's theorem is

> Theorem. At any step j of the block Lanczos algorithm and for all i
>
> $$|y_i^* P_{j+1} B_{j+1} s_{ji}| = \gamma_{ji} \varepsilon \|A\| ,$$
>
> with $\gamma_{ji} \simeq 1$ and s_{ji} is the last m elements of s_i .

This shows that the quantity $B_{j+1}s_{ji}$ is again the key factor in both accuracy and loss of orthogonality but it now takes two different interpretations. The residual norm of $(y_i^{(j)}, \theta_i^{(j)})$ is $\|B_{j+1}s_{ji}\|$ but orthogonality between y and some vector in P_{j+1} is lost if the corresponding element of $B_{j+1}s_{ji}$ is small. We define $\beta_{ji} = \|B_{j+1}s_{ji}\|$ as before and let ω_{ji} be the absolute value of the smallest element of $B_{j+1}s_{ji}$. Thus

Convergence => loss of orthogonality

but not vice versa. For example, let m = 2, q_1 be arbitrary and $R_0 = (q_1, Aq_1)$. Then since

$$\text{span}(R_0, AR_0) = \text{span}(q_1, Aq_1, Aq_1, A^2q_1)$$

has dimension 3, R_1 will be rank 1. Numerically R_1 will not be exactly rank 1 but the severe cancellation which occurs in its calculation will cause drastic loss of orthogonality even though no convergence has occurred. Despite this distinction it is still straight forward to implement selective orthogonalization in the block Lanczos context.

So far we have discussed the two factors which favor the choice of a large block size. However, there are two other factors which favor a small block size. The cost of computing eigenvalues of T_j increases as a quadratic function of the blocksize and the number of vector inner products of length n needed at each step of the algorithm is linear in m. Furthermore, the asymptotic convergence of the algorithm depends on the number of steps taken before the algorithm must be iterated. So for difficult problems with limited storage available it is important to take a small block size in order to maximize the number of steps taken in each iteration.

2. MAJOR DESIGN DECISIONS

2.1 STANDARDIZATION AND MODULARIZATION

It was decided to adhere strictly to the 1966 FORTRAN standard to help assure portability. This standardization was checked by running the codes through the PFORT verifier [11]. It was also decided to avoid the use of private common blocks and the use of the EQUIVALENCE statement completely.

It was decided to make use of existing high quality software for three reasons: to make the codes more efficient, to improve modularity of the codes, and to improve readability of the source code. EISPACK subroutines

(Smith et al. [13] and Garbow et al. [4]) were used for all subsidiary eigenvalue calculations on dense matrices, a subset of the basic linear algebra subprograms (BLAS) (Lawson et al. [5] and Dongarra et al. [2]) was used for all vector manipualtion, and URAND (Forsythe et al. [3]) was used for uniform random number generation.

2.2 PROBLEM DEFINITION

The two major questions to be resolved are:

1. Are eigenvalues to be computed at both ends of the spectrum of A or only one?
2. Are a fixed number of eigenvalues to be computed or all the eigenvalues outside some boundary to be found?

At first glance, one end only is the answer to question 1. In almost all applications, A will be the discretation of some continuous operator on a function space. One end of the spectrum of A will be an accurate representation of the spectrum of the original operator while the other end will be discretation noise.

However, one of the more powerful techniques of eigenvalue extraction is sectioning, in which a few eigenvalues of

$$(A - \sigma I)^{-1}$$

are found for various values of the shift σ. The inversion of the spectrum changes a problem of finding all the eigenvalues of A inside a given interval to finding all the eigenvalues of $(A - \sigma I)^{-1}$ outside a given (different) interval. For this reason it was decided to allow two types of problems.

1. Find some eigenvalues at one end of the spectrum.
2. Find all the eigenvalues outside a given interval.

It was decided to specify a fixed number of eigenvalues in case 1. The problem of finding all the eigenvalues of A smaller than some number α can be handled (with only minor inefficiencies) by case 2 using a very large number for the right end point of the excluded interval.

2.3 MATRIX-VECTOR PRODUCTS

The Lanczos algorithm requires the formation of matrix-vector products. There are two possible approaches to supplying this information. Either a user supplied subroutine is needed or reverse communication is used, in which control is returned to the calling program, for each matrix multiply.

Four reasons favored the use of a user supplied subroutine. The matrix multiply is needed in the inner loop of the algorithm which is likely to be buried several layers deep in the subroutine structure of the code. To return to the main program each time would significantly increase the number of subroutine linkages required at execution time.

Reverse communication would also violate the FORTRAN standard since the standard does not require that local variables in subroutines maintain their values between calls. Furthermore reverse communication, which allows the user to change parameter values is much more prone to accidental contamination.

Finally, the source code using a user supplied subroutine is much cleaner and easier to read since there is no need to monitor different types of entry conditions.

2.4 STORAGE OF LANCZOS VECTORS

There are three approaches to storage of the Lanczos vectors (q's). One approach is to require that they be stored in core (either in common or in workspace provided in the calling sequence). This method was rejected

immediately since the main point of selective orthogonalization is to maintain robust linear independence among the Lanczos vectors without having to keep them in core.

Another possibility is to require the user to supply a logical unit number for storing Lanczos vectors and store them using unformatted WRITE statements. The disadvantage of this approach is that it forces the vectors to be stored on disk even if sufficient core exists (as might easily be true for a machine with virtual memory).

The approach actually chosen was for the user to supply a subroutine for storing and recalling Lanczos vectors. This allows greater flexibility at the cost of extra subroutine linkages.

2.5 STORAGE OF RITZ VECTORS

Is is feasible for the Ritz vectors (y's) to be put into secondary store? The answer is a strongly qualified yes, provided that sufficient core storage exists for one Ritz vector. However, the overhead involved is quite high. Each time a Ritz vector is needed for orthogonalization it would have to be recalled from disk. Even more important, the formation of each single Ritz vector would require recalling all the Lanczos vectors, instead of forming all the Ritz vectors needed simultaneously with one pass through the Lanczos vectors. Finally, it would be impossible to assure that the computed Ritz vectors were orthogonal to working accuracy.

For these reasons, it was decided to require that the desired Ritz vectors be kept in fast memory.

2.6 LIMITS AND RECALL CAPABILITY

Eigenvalue extraction is inevitably an iterative process. Therefore it is necessary to provide some external stopping criterion to avoid the

possibility of an infinite loop. This is especially true in the Lanczos context where an error in the matrix multiply routine will almost always prevent the algorithm from converging.

It was decided to use the number of calls to the matrix multiply subroutine, OP, as the stopping criterion. However, since it is impossible for the code to determine a priori a "reasonable" value for this limit, it is necessary for the user to set this limit on input. If the limit on the number of calls to OP is reached the program terminates, returning all the eigenpairs which have been determined and resetting the parameters so that the subroutine can be immediately recalled to continue working on the problem.

This restart capability allows a user who knows one (or more) eigenpairs of the matrix to supply this information so that the code does not expend effort recomputing known eigenpairs. However, it should be noted that any such user supplied eigenpairs are counted as desired eigenpairs as far as solving the given problem.

2.7 LOCAL REORTHOGONALIZATION

Several authors including Lewis [6] and Ruhe [10] recommend the use of local reorthogonalization in which the latest block of Lanczos vectors are reorthogonalized against the two previous ones (which are still in core). Since the cost is small relative to the cost of a matrix multiply, it was decided to use local reorthogonalization.

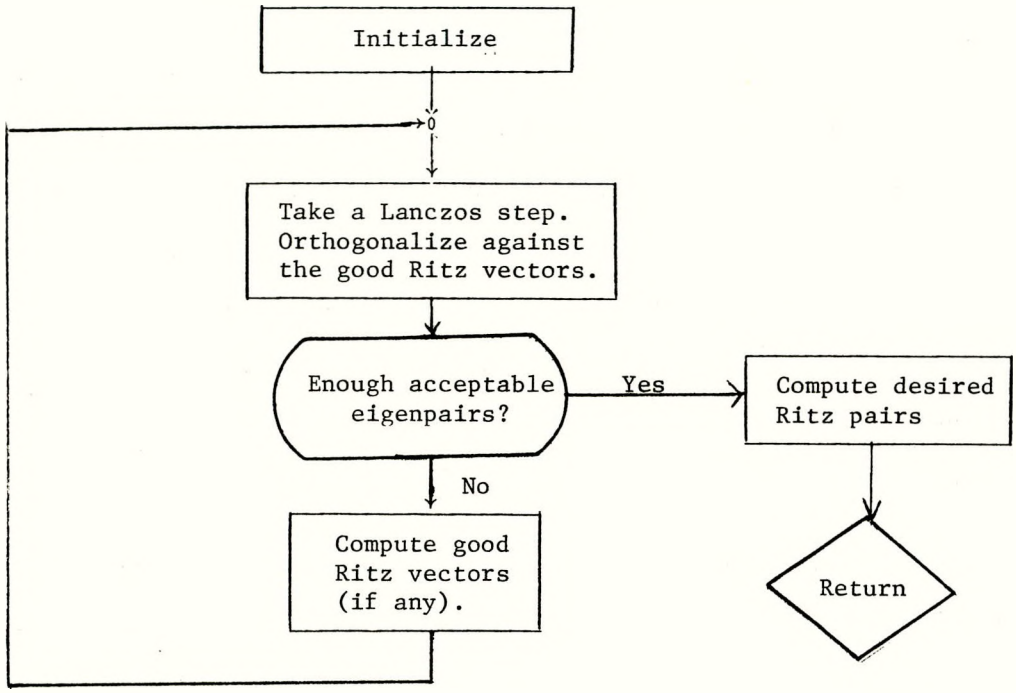
3. ALGORITHMIC DETAILS

The programs basic purpose is to implement the block Lanczos algorithm with selective orthogonalization and to stop as soon as all the desired

eigenvalues are sufficiently accurate (acceptable). Let ω_{ji} be defined as the absolute value of the smallest element of the vector $B_{j+1}s_{ji}$ (see Section 1). Then the Ritz vector $y_j^{(j)}$ is good whenever $\omega_{ji} \leq \sqrt{\varepsilon}\|A\|$ and the next Lanczos vector should be orthogonalized against $y_j^{(j)}$.

3.1 FLOWCHART 1

A naive flowchart of the algorithm would be as follows.



The above algorithm is inefficient in three respects and incomplete in several more as we describe below.

3.2 INEFFICIENCIES IN FLOWCHART 1

In theory all the Ritz vectors change at every Lanczos step and so the good Ritz vectors should be recomputed at every step. This would be ruinously expensive. Fortunately, the good Ritz vectors change very little from one step to the next and so a good Ritz vector computed at some earlier step can be used for orthogonalization at the current step.

Furthermore it is not necessary to orthogonalize against a good Ritz vector at every step. In Parlett and Scott [9] it is shown that if (y,θ) is a good Ritz pair and $|y^*q_{j+1}| \leq \tau_{j-1}$ and $|y^*q_j| \leq \tau_j$ then

$$|y^*q_{j+1}| \leq [|\theta - \alpha_j|\tau_j + \beta_j\tau_{j-1}]/\beta_{j+1} \equiv \tau_{j+1}$$

except for terms of order ε.

For blocksize greater than 1 the above formula must be modified. If $\|y^*P_{j-1}\| \leq \tau_{j-1}$ and $\|y^*P_j\| \leq \tau_j$ then

$$\|y^*P_{j+1}\| \leq [\|\theta - A_j\|\tau_j + \sigma_j^L\tau_{j-1}]/\sigma_{j+1}^S \equiv \tau_{j+1}$$

where σ_j^L is the largest singular value of B_j and σ_{j+1}^S is the smallest singular value of B_{j+1}. Whenever $\tau_{j+1} > \sqrt{\varepsilon}$, P_{j+1} and P_{j+2} are orthogonalized against y and τ_{j+1} and τ_{j+2} are set to ε .

Finally, it is not necessary to compute all the desired eigenvalues of T at each step. In general the Lanczos algorithm will converge faster to the extreme eigenvalues of the spectrum of A,i.e. convergence will occur monotonically inward from the edge of the spectrum. Therefore for a number type

problem it is necessary to compute only 2 eigenpairs at most steps. First the most interior desired eigenpair is computed to see if it is acceptable. If it is acceptable, then all the desired eigenpairs are computed to insure that they are all acceptable. If not all the eigenvalues are acceptable, then the most extreme eigenpair which has not been declared good is computed. If it is still not good, then the next Lanczos step is taken. If it is good then all of the desired eigenpairs are computed and all the good Ritz vectors are computed. It seems reasonable to recompute known good Ritz vectors for two reasons. First the eigenpairs continue to converge so that the newer version of a good Ritz vector is more accurate than the old one. Secondly the major cost of forming the Ritz vectors is in recalling the sequence of Lanczos vectors. Since this must be done anyway for the new good Ritz vectors there is little extra cost in updating the old ones as well.

For an interval type problem no more than six eigenvalues of T are computed at one step. The details are left for a later chapter.

3.3 LIMITED STORAGE

Since the available storage on a computer is limited, it may not be possible to take another Lanczos step despite the fact that not all the desired eigenvalues are acceptable. The computer code must allow the user to specify this limit. If the limit is reached the code computes and stores all acceptable eigenpairs (we refer to such vectors as permanent vectors) and then forms a new starting block from linear combinations of the remaining eigenvectors. The Lanczos algorithm is then restarted with this starting block.

This restarting of the algorithm requires one additional precaution. The new Lanczos sequence must be kept orthogonal to the permanent vectors. As before it is not necessary to orthogonalize against the permanent vectors

at each step. Instead a scalar recurrence is used to monitor the decay of orthogonality. The recurrence for permanent vectors is the same as the recurrence for good vectors except that the term ρ/σ^{S}_{j+1} must be added to τ_{j+1}, where ρ is the residual norm of the permanent vector y ($\|Ay - y\theta\| \simeq \beta_{ji}$).

It should be noted that restarting the algorithm is always detrimental to the convergence rate of the algorithm. So that provided the user subroutines are known to be reliable, all the available storage should be used.

3.4 MULTIPLE EIGENVALUES

As mentioned in Chapter 1, the block Lanczos algorithm cannot find more than m (the blocksize) eigenvectors associated with a multiple eigenvalue. For numerical purposes this means that if the code computes m eigenvalues which are tightly clustered then it is possible that there are more eigenvalues in the cluster. After the code has computed the desired eigenvalues it checks to see if a cluster of m eigenvalues exists. If such a cluster is found the algorithm is restarted just as if not all the desired eigenpairs were acceptable, except that random vectors are used for the starting block. This causes several complications. The codes must now be able to terminate in the middle of a Lanczos run when it decides that no new eigenvalues exist in the cluster. Furthermore, if solving a number type problem (see Section 2.2) the code must throw away a permanent vector if a new member of the cluster is found.

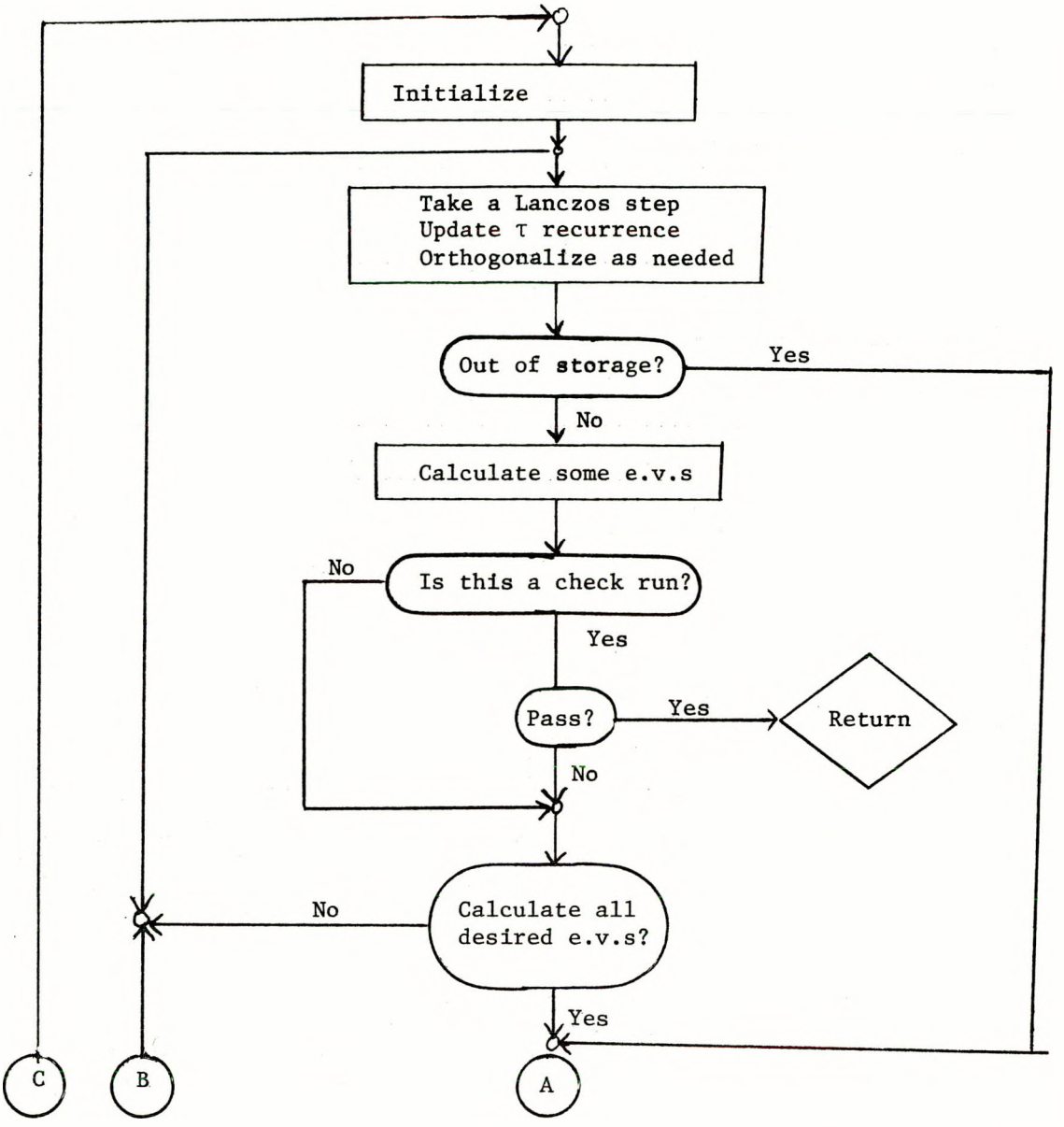
For example, suppose that the five smallest eigenvalues of A are 0,0,0,1, and 2. If the three smallest eigenvalues are desired and a blocksize m = 2 is used. Then the eigenvalues 0,0, and 1 will be found by the first Lanczos sequence. Since a multiplicity of 2 exists in the determined eigenvalues

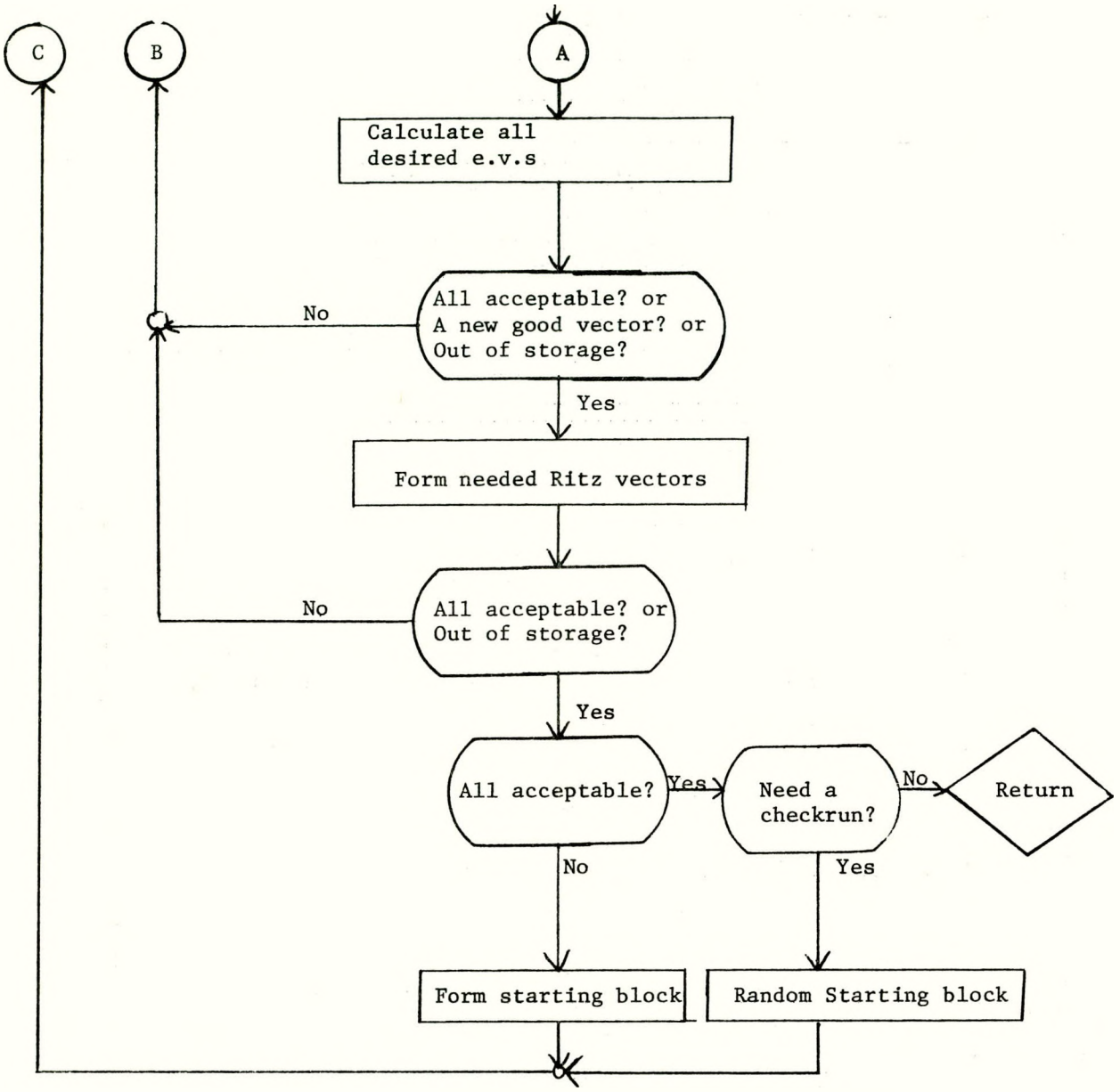
a check run would be started. Soon after, the third copy of zero would be found and the eigenvalue 1 (and its eigenvector) would be discarded to make room for it. On the other hand if the zero eigenvalue were only double, the check run would terminate after some number of steps and the values 0,0, and 1 would be returned.

From the point of view of efficiency, it is desirable that the block size be larger than the largest multiplicity of the desired eigenvalues, so that no check run need be made. In particular a block size of one is recommended only for number type problems and only when one eigenvalue is desired. (Of course storage constraints may require the use of a block size one.)

3.5 FLOWCHART II

The following flowchart reflects the results of the previous sections. The abbreviation e.v.s. stands for eigenvalues (of T).


Initialize
Take a Lanczos step
Update τ recurrence
Orthogonalize as needed
Out of storage?
Yes
No
Calculate some e.v.s
No
Is this a check run?
Yes
Pass?
Yes
Return
No
No
Calculate all
desired e.v.s?
Yes
C
B
A



This flowchart is a fairly accurate representation of the two codes. The specific details of the implementations will be discussed in the sections covering the individual subroutines. However, there are two more topics of sufficient generality to be discussed here.

3.6 FORMING THE STARTING BLOCK

If storage has been exhausted, it is necessary to compute m vectors to form the new starting block. The simplest approach is to take the first m desired eigenvectors which are not acceptable. However, if there are more than m unacceptable desired vectors, this means that some eigenvectors are completely ignored. At best this is a waste of valuable information and at worst it can lead to the algorithm failing to find some eigenvalues.

On the other hand to simply sum up all the unacceptable vectors is also ineffective since it is possible for a fairly accurate Ritz vector to be swamped by one or more completely inaccurate Ritz vectors. To avoid these two extremes the code has all the unacceptable Ritz vectors contribute to the starting block but weights the contributions by the inverse of the residual norm of the vector. Thus almost acceptable vectors will have large weights compared to very poor vectors.

Finally, it should be noted that the unacceptable Ritz vectors are not calculated individually. Rather the appropriate linear combinations of eigenvectors of T are formed and these are used to form the starting block directly.

3.7 ORDERING THE INNER LOOP

It turns out to be better to rearrange the inner loop so that the orthogonalizations occur at the top, as follows:

Given $P_0 \equiv 0$, $R_0 = P_1'$ and $B_1' = I$. For $j = 1,2,\ldots$

1. Orthogonalize P_j' against any vector indicated by the τ recurrence.
2. If necessary (j = 1 or some orthogonalization done) reorthognoralize so that

$$P_j = P_j' \hat{B}_j = R_{j-1} B_j' \hat{B}_j$$

and set $B_j = B'_j \hat{B}_j$

3. $U_j = AP_j - P_{j-1}B^*_j$

4. $A_j = P^*_j AP_j$

5. $R_j = U_j - P_j A_j$

6. $R_j = P'_{j+1} B'_{j+1}$ with P'_{j+1} orthonormal

7. Update the τ relations .

We do not actually need the orthonormality of P'_{j+1} until the following Lanczos step but we do need the matrix B'_{j+1} in order to compute β_{ji} and ω_{ji} (the residual norm and the orthogonality coefficient) for the eigenvalues we compute.

This ordering has several advantages. Firstly, we do not have to do any special coding (other than initializing the τ recurrence properly) to handle the first step. The initial block is naturally orthogonalized against the known vectors (if any) and then orthonormalized). Furthermore the decisions about terminating, restarting, or computing new good Ritz vectors are made before any unnecessary orthogonalizations are performed.

One final note: throughout this report we have used the subscript j to denote the step number. In the code we found it more convenient to use J as the dimension of T_j, that is, J = j*m .

4. SUBROUTINE STRUCTURE

4.1 INTRODUCTION

Implementing the Lanczos algorithm requires temporary storage space for a variety of purposes. Unfortunately FORTRAN does not allow EQUIVALENCE to be used with formal parameters. For this reason it is impossible to break

the allotted work space into different named areas inside the first subroutine called. There are three possible ways to circumvent this problem.

One approach is to require the user to break up the workspace in the original calling sequence by having a number of separate workspace parameters. This method was rejected because it greatly increases the probability of user errors in calling the subroutine. Another approach is to use various integer variables to keep track of the offsets for different work space areas. This method makes the corresponding code quite intricate (especially when following the standard restrictions on the form of array subscripts) and almost indecipherable.

Instead it was decided to have the driver subroutine be almost a dummy routine which did little more than call other routines to do most of the work. Thus the available workspace could be subdivided in the calling sequences to these subsidiary subroutines.

In the end, two such subroutines were written. The major one implements the block Lanczos algorithm with selective orthogonalization to solve the given problem and the second one post-processes the eigenvectors that were computed by the first.

In addition a number of additional functions and subroutines are used for specific computational tasks. Three of these tertiary subroutines were written by the author, while the rest were taken from published sources.

It turned out to be inconvenient to solve both types of problems described in Section 2.1 with the same subroutine. Two separate driver subroutines were written for the two different problem types. This in turn required separate subsidiary subroutines, but the specific computational modules (tertiary subroutines) are the same for both problem types.

4.2 SUBROUTINE NAMING CONVENTIONS

Each of the major subroutines has a root name, a letter preceding the root name which identifies which of the two problem types it is used for, preceded by a letter indicating the precision employed. The codes are

LASO	root name, driver subroutine.
PPLA	root name, post processor.
WLA	root name, main computation subroutine.
N	identifier, problem type 1 (some number of eigenvalues at one end of the spectrum).
I	identifier, problem type 2 (all the eigenvalues outside a given interval).
S	single precision.
D	double precision.

Thus SNLASO is the single precision driver program for a number type problem, while DIPPLA is the double precision post-processor for an interval type problem.

The tertiary subroutines have naming conventions which depend on their source. EISPACK subroutine names are fixed, regardless of whether they are in single or double precision. The basic linear algebra subroutines (BLAS) in LINPACK have a root name preceded by an S or a D to indicate the level of precision. The tertiary subroutines written by the author also have a root name preceded by an S or a D. Finally URAND, taken from [3] exists only in single precision since double precision integers are not provided for in the 1966 FORTRAN standard.

4.3 SUBROUTINE GLOSSARY

BANDR: EISPACK subroutine for reducing a symmetric band matrix to tridiagonal form.

BANDV: EISPACK subroutine for computing eigenvectors of a band matrix by inverse iteration.

BISECT: EISPACK subroutine for computing all the eigenvalues of a symmetric tridiagonal matrix inside a given interval. Used only in interval type problems.

DAXPY: double precision BLAS subroutine for adding a scalar multiple of one vector to another.

DCOPY: double precision BLAS subroutine for copying one vector into another.

DDOT: double precision BLAS function for computing a vector inner product.

DILASO: double precision driver for interval type problems.

DIPPLA: double precision post processor for interval type problems.

DIWLA: double precision computation subroutine for interval type problems.

DMVPC: double precision subroutine for computing the residual norm and orthogonality coefficient for given Ritz pairs.

DNLASO: double precision driver for number type problems.

DNPPLA: double precision post processor for number type problems.

DNRM2: double precision BLAS function for computing a vector norm.

DNWLA: double precision computation subroutine for number type problems.

DORTQR: double precision subroutine for orthonormalizing a set of vectors.

DSCAL: double precision BLAS subroutine for multiplying a vector by a scalar.

DSWAP: double precision BLAS subroutine for swapping two vectors.

DVZERO: double precision subroutine for zeroing a given vector.

IMTQL1: EISPACK subroutine for computing all the eigenvalues of a symmetric tridiagonal matrix.

IMTQL2: EISPACK subroutine for computing all the eigenvalues and eigenvectors of a symmetric tridiagonal matrix.

SAXPY: single precision BLAS subroutine for adding a scalar multiple of one vector to another.

SCOPY: single precision BLAS subroutine for copying one vector onto another.

SDOT: single precision BLAS function for computing a vector inner product.

SILASO: single precision driver for interval type problems.

SIPPLA: single precision post processor for interval type problems.

SIWLA: single precision computation subroutine for interval type problems.

SMVPC: single precision subroutine for computing the residual norm and orthogonality coefficient for given Ritz pairs.

SNLASO: single precision driver for number type problems.

SNPPLA: single precision post processor for number type problems.

SNRM2: single precision BLAS function for computing a vector norm.

SNWLA: single precision computation subroutine for number type problems.

SORTQR: single precision subroutine for orthonormalizing a set of vectors.

SSCAL: single precision BLAS subroutine for multiplying a vector by a scalar.

SSWAP: single precision BLAS subroutine for swapping two vectors.

SVD: EISPACK subroutine for computing the singular values of a matrix.

SVZERO: single precision subroutine for zeroing a vector.

TRED1: EISPACK subroutine for reducing a symmetric matrix to tridiagonal form. The transformations are not accumulated.

TRED2: EISPACK subroutine for reducing a symmetric matrix to tridiagonal form. The transformations are accumulated.

URAND: a single precision function for generating uniform random numbers on the interval [0,1].

5. TERTIARY SUBROUTINES

The EISPACK subroutines used (BANDR, BANDV, BISECT, IMTQL1, IMTQL2, SVD, TRED1, and TRED2) are documented in Smith et al. [13] and B. S. Barbow it al. [4]. The Basic Linear Algebra Subroutines are taken from LINPACK which is documented in Dongarra et al. [2]. URAND is documented in Forsythe et al. [3]. The three remaining routines were written by the author and will be documented here. Only the single precision versions will be explicitly documented. The arguments of each subroutine are classified according to the following table.

Table 1. Argument Classifications

SI	Strict input: The original value is always referenced.
I	Input: The original value is sometimes referenced.
SO	Strict output: A new value is always returned.
O	Output: A new value is sometimes returned.
W	Workspace: The original value is never referenced and the final value is meaningless.

5.1 SMVPC

SMVPC is used to compute the residual norm estimate β_{ji} and the orthogonality coefficient ω_{ji} (see Chapter 1) for one or more eigenvectors of T_j. β_{ji} is used in determining the accuracy of the corresponding eigenvalue while ω_{ji} is used to monitor the loss of orthogonality. Since the major cost of computing either ω_{ji} or β_{ji} is in forming the product $B_{j+1}s_{ji}$, SMVPC always returns both values.

5.1.1 The Calling Sequence

SUBROUTINE SMVPC(NBLOCK, BET, MAXJ, J, S, NUMBER, RESNRM, ORTHCF, RV)

Table 2. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
NBLOCK	integer	scalar	SI	Blocksize.
BET	real	NBLOCK*NBLOCK	SI	B_{j+1}.
MAXJ	integer	scalar	SI	Leading dimension of S.
J	integer	scalar	SI	Dimension of the eigenvectors of T_j.

Table 2. (Cont'd)

Name	Type	Dimension	Classification	Remarks
S	real	MAXJ*NUMBER	SI	The eigenvector(s) of T_j.
NUMBER	integer	scalar	SI	The number of eigenvector(s).
RESNRM	real	NUMBER	SO	The β_{ji}.
ORTHCF	real	NUMBER	SO	The ω_{ji}.
RV	real	NBLOCK	W	Workspace for forming $B_{j+1}s_{ji}$.

5.1.2 Internal Variables

The subroutine uses three internal variables:

Table 3. Internal Variables

Name	Type	Remarks
I	integer	DO loop index over the eigenvectors (1 to NUMBER).
K	integer	DO loop index over the rows of B_{j+1} (1 to NBLOCK).
M	integer	Subscript, set to J - NBLOCK + 1. S(M,I) is the first element of s_{ji}.

5.1.3 Sequence Blocks

In reference to the listing of SMVPC in appendix 2.

Table 4. Sequence Blocks

Sequence Numbers	Remarks
10-200	Initial declarations and comments.
210	Set M = J-NBLOCK + 1.
220-340	DO loop in I over number of vectors.

Table 4. (Cont'd)

Sequence Numbers	Remarks
250-290	DO loop in K over rows of BET. Forms matrix vector product and updates minimum element (ORTHCF (I)).
310	Computes RESNRM (I) = β_{ji} .
320	Scales ORTHCF (I).
350-370	Exit.

5.1.4 Special Note

The orthogonality coefficients do not reflect the use of local reorthogonalization. This will cause vectors to be declared good somewhat sooner than necessary. This is particularly evident when the starting block contains a good approximation to an eigenvector. Then cancellation will cause loss of orthogonality to the second block but reorthogonalization of the second block to the first corrects the problem. To account for the reorthogonalization against the first block we found it advantageous to scale the orthogonality coefficient ω_{ji} by a factor $\|\hat{s}_i\|^{-1}$ where $\hat{s}_i$ is the J - 2*NBLOCK vector obtained from s_i by deleting the top 2*NBLOCK elements. This always delays the classification of a good vector and if the corresponding eigenvector of T has most of its mass concentrated in the first few components (as is the case if the starting block contains a near eigenvector) this delay is significant.

5.2 SORTQR

SORTQR is used to orthonormalize a block of Lanczos vectors. This simplest (cheapest) approach is to use modified Gram-Schmidt. Unfortunately

this requires either pivoting or reorthogonalization for stability. If pivoting is used the corresponding matrix of coefficients is not triangular and this increases the band width of the matrix T.

So following Lewis [6] we decided to use householder transformations to orthogonalize the blocks. Indeed SORTQR is essentially Lewis' QRDECM modified to incorporate the Basic Linear Algebra Subprograms.

5.2.1 The Calling Sequence

Table 5. SUBROUTINE SORTQR(N,NBLOCK,Z,B)

Name	Type	Dimension	Classification	Remarks
N	integer	scalar	SI	Length of vectors.
NBLOCK	integer	scalar	SI	Number of vectors.
Z	real	N*NBLOCK	SI-SO	The block of vectors on input and the orthonormalized vectors on output.
B	real	NBLOCK* NBLOCK	SO	The upper triangular matrix.

5.2.2 Internal Variables

Table 6. Internal Variables

Name	Type	Remarks
I	integer	Index over the columns of Z (one to NBLOCK in reduction phase and NBLOCK to one in the accumulation phase.)
J	integer	Set equal to I + 1 to index a DO loop (twice).
LENGTH	integer	Set equal to the length (dimension) of the current Householder reflection.
M	integer	Used as the DO loop variable when I is counting downward.

Table 6. (Cont'd)

Name	Type	Remarks
SIGMA	real	Set to the norm (with sign chosen for stability) of the current vector used for forming the Householder reflection.
TAU	real	Set to the normalizing factor for the current reflection.
TEMP	real	Used to hold the multiplier for the following call to SAXPY. Not strictly needed but included for readability.

5.2.3 Sequence Blocks

In reference to the listing of SORTQR in appendix 2:

Table 7. Sequence Blocks

Sequence Numbers	Remarks
10-180	Initial comments and Declarations.
190-410	DO loop in I over number of vectors.
230-290	Form $I^{\underline{th}}$ reflections.
330-390	DO loop in K applying the reflection to the rest of the vectors.
450-660	DO loop in M over the number of vectors as I runs backwards through the vectors.
490-550	Recreate $I^{\underline{th}}$ reflection.
590-620	DO loop in K applying this reflection to the previously accumulated reflections.
640-650	Construct current column.
670-690	Exit.

5.3 SVZERO

SVZERO is used to initialize a given vector to zero. The structure of the subroutine is copied from SCOPY except that the increment is known to be one.

5.3.1 The Calling Sequence

SUBROUTINE SVZERO (N,Q)

Table 8. SUBROUTINE SVZERO (N,Q)

Name	Type	Dimension	Classification	Remarks
N	integer	scalar	SI	Length of vector.
Q	real	N	SO	The vector.

5.3.2 Internal Variables

Table 9. Internal Variables (M,MP1,I)

Name	Type	Remarks
M	integer	Set to N mod 7.
MP1	integer	Set to M + 1.
I	integer	DO loop variable (1 to M and MP1 to N by 7).

5.3.3 Sequence Blocks

In reference to the listing of SVZERO in appendix 2:

Table 10. Sequence Blocks

Sequence Numbers	Remarks
10-130	Initial comments and declarations.
140-210	Clean up loop.
220-320	Main loop.
330-350	Exit.

6. SNLASO

SNLASO is the driver program for number type problems. It checks the consistency of the calling parameters, orthonormalizes any user supplied eigenvectors, and calls the subroutines SNWLA and SNPPLA.

6.1 THE CALLING SEQUENCE

SUBROUTINE SNLASO (OP,IOVECT,N,NVAL,NFIG,NPERM,NMVAL,VAL,NMVEC,VEC, NBLOCK,MAXOP,MAXJ,WORK,IND,IERR) .

Table 11. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
OP	external	-	I	User supplied subroutine for matrix-vector products.
IOVECT	external		I	User supplied subroutine for storing and recalling vectors.
N	integer	scalar	SI	The dimension of the matrix.
NVAL	integer	scalar	SI	Indicates the number of desired eigenvalues.

Table 11. (Cont'd)

Name	Type	Dimension	Classification	Remarks
NFIG	integer	scalar	SI	The number of decimal digits of accuracy desired.
NPERM	integer	scalar	SI-SO	The number of eigenvectors known.
NMVAL	integer	scalar	SI	The row dimension of VAL.
VAL	real	NMVAL*4	I-O	The eigenvalues and accuracy estimates.
NMVEC	integer	scalar	SI	The row dimension of VEC.
VEC	real	NMVEC*NVAL	I-O	The eigenvectors.
NBLOCK	integer	scalar	SI	The block size (number of vectors per block).
MAXOP	integer	scalar	SI	The maximum number of calls to OP.
MAXJ	integer	scalar	SI	The limit on the number of vectors stored by IOVECT. It also effects WORK.
WORK	real	(see below)	I-O	Workspace. The first N*NBLOCK elements are the starting block.
IND	integer	NVAL	SO	Used for workspace and to return the actual number of calls to OP.
IERR	integer	scalar	SO	An error completion code.

The array WORK must be at least as long as

NBLOCK*(3*N + 2*NBLOCK) + MAXJ*(3*NBLOCK + ABS(NVAL) + 6)

+ 3*ABS(NVAL).

For a more detailed description of the calling sequence see Appendix 1.

6.2 INTERNAL VARIABLES

Table 12. Internal Variables

Name	Type	Remarks
DELTA	real	Returned from SNWLA as the eigenvalue of A closest to the desired eigenvalues. Used in SNPPLA for computing the accuracy estimates.
I	integer	Used as the primary DO loop index.
I1-I14	integer	Used as subscripts in the calls to SNWLA and SNPPLA.
K	integer	Used as the secondary DO loop index.
M	integer	Used as a DO loop limit and as an array subscript.
NOP	integer	Returned from SNWLA and SNPPLA as the actual number of calls to OP. Stored in IND(1) just before exit.
NV	integer	Set to abs(NVAL).
RARITZ	logical	Returned from SNWLA and passed to SNPPLA. RARITZ is .TRUE. if a final Rayleigh-Ritz procedure is needed.
SMALL	logical	Set to .TRUE. if the leftmost eigenvalues are desired. Passed to SNWLA and SNPPLA.
TEMP	real	Used for temporary storage for sorting etc.

6.3 SEQUENCE BLOCKS

In reference to the listing of SNLASO in Appendix 2:

Table 13. Sequence Blocks

Sequence Numbers	Remarks
10-1860	Initial comments and declarations.
1870-2070	Check consistency and set local parameters.

Table 13. (Cont'd)

Sequence Numbers	Remarks
2080-2810	Orthonormalize user supplied eigenvectors.
2820-3100	Call SNWLA.
3110-3220	Call SNPPLA.
3230-3250	Set IND(1) and exit.

7. SNWLA

SNWLA is the subroutine which implements the block Lanczos algorithm with selective orthogonalization to compute the desired eigenvalues. If the largest (rightmost) eigenvalues are desired (SMALL = .FALSE.) the code implicitly uses the negative of the matrix (by negating the matrix T) and otherwise always computes the smallest eigenvalues.

7.1 THE CALLING SEQUENCE

SUBROUTINE SNWLA(OP,IOVECT,N,NVAL,NFIG,NPERM,VAL,NMVEC,VEC,NBLOCK,MAXOP, MAXJ,NOP,PO,P1,P2,RES,TAU,OTAU,T,ALPHA,BETA,BETA2,ALP,BET,RV,RV6,S,IND,SMALL, RARITZ,DELTA,IERR).

Table 14. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
OP	external	-	SI	Forms matrix vector products.
IOVECT	external	-	SI	Used to store and recall vectors.
N	integer	scalar	SI	Dimension of the matrix.
NVAL	integer	scalar	SI	The number of eigenvalues desired (positive).

Table 14. (Cont'd)

Name	Type	Dimension	Classification	Remarks
NFIG	integer	scalar	SI	The number of decimal digits of accuracy desired in the eigenvalues.
NPERM	integer	scalar	SI-O	On input, the number of user supplied eigenpairs. On output the number of eigenpairs now known (usually NVAL). In between, the number of permanent vectors.
VAL	real	NVAL	I-O	On input, the user specified eigenvalues. Internally, the permanent eigenvalues, followed by the good eigenvalues, followed by the eigenvalues computed at the current step. On output, all the known eigenvalues.
NMVEC	integer	scalar	SI	The row dimension of VEC.
VEC	real	NMVEC*NVAL	I-O	On input, the user supplied eigenvectors. Internally the permanent eigenvectors followed by the good eigenvectors. On output, all the known eigenvectors.
NBLOCK	integer	scalar	SI	The number of vectors in each Lanczos block.

Table 14. (Cont'd)

Name	Type	Dimension	Classification	Remarks
MAXOP	integer	scalar	SI	The maximum number of calls to OP. SNWLA is aborted (IERR = -2) if this maximum is reached. Note that the comparison is turned off if the code is making a check run to test for multiplicities so that the number of calls to OP may be larger than MAXOP before termination of SNWLA.
MAXJ	integer	scalar	SI	The maximum number of Lanczos vectors which can be stored. The algorithm is restarted (iterated) if this maximum is reached.
NOP	integer	scalar	SO	The number of calls to OP.
P0	real	N*NBLOCK	W	The "oldest" block of Lanczos vectors kept in fast storage.
P1	real	N*NBLOCK	SI	The "middle" block of Lanczos vectors. On input P1 must contain the desired starting vectors. Zero vectors are replaced by random vectors. On output, if IERR = -2, P1 will contain the best vectors for restarting the algorithm.

Table 14. (Cont'd)

Name	Type	Dimension	Classification	Remarks
P2	real	N*NBLOCK	W	The "newest" block of Lanczos vectors.
RES	real	NVAL	W	Holds the residual norms of the permanent vectors. Needed to update the τ recurrence.
TAU	real	NVAL	W	Holds the current value of τ for each eigenvector.
OTAU	real	NVAL	W	Holds the previous value for τ for for each eigenvector. Note that whenever an orthogonalization is performed it is necessary to orthogonalize <u>two</u> successive blocks against the indicated eigenvector. Rather than keep a separate pointer the code sets TAU = 0 and OTAU = 1 the first time an orthogonalization is performed. Since OTAU = 1 the TAU at the next step will be bigger than $\sqrt{\epsilon}$ so another orthogonalization is performed. However the fact that TAU = 0 is used to suppress a third orthogonalization by not setting OTAU = 1 again.

Table 14. (Cont'd)

Name	Type	Dimension	Classification	Remarks
T	real	MAXJ* (NBLOCK+1)	W	Holds the band matrix.
ALPHA	real	MAXJ	W	Holds the diagonal elements of T reduced to tridiagonal form (by a call to BANDR).
BETA	real	MAXJ	W	Holds the off diagonal elements of T reduced to tridiagonal form.
BETA2	real	MAXJ	W	Holds the squares of the off diagonal elements.
ALP	real	NBLOCK*NBLOCK	W	Holds the lower triangle of the current diagonal block of the band matrix T. The full square of storage is used for ease of addressing.
BET	real	NBLOCK*NBLOCK	W	Holds B_{j+1} the next off diagonal block of T (which is actually upper triangular).
RV	real	MAXJ* (2*NBLOCK+1)	W	Used as workspace for various subroutines. Only BANDV requires the full array. The first column of RV contains the orthogonality coefficients on exit from SMVPC.

Table 14 (Cont'd)

Name	Type	Dimension	Classification	Remarks
RV6	real	MAXJ	W	Used as workspace for various subroutines. RV6 contains the residual norms (β_{ji}) on exit from SMVPC.
S	real	MAXJ*NVAL	W	Used to hold eigenvectors of T on exit from BANDV. Also used to manipulate the eigenvectors of T to form the appropriate linear combination needed in computing the new starting vectors.
IND	integer	NVAL	W	Used in various EISPACK subroutines. Also used for pointers in various sorting operations.
SMALL	logical	scalar	SI	SMALL = .TRUE. if the smallest (leftmost) eigenvalues are desired. If SMALL = .FALSE. then the matrix T is negated.
RARITZ	logical	scalar	SO	Set to .TRUE. if a Rayleigh-Ritz procedure is to be done in SNPPLA. This happens if a cluster of more than NBLOCK eigenvalues are found.
DELTA	real	scalar	SO	Set to the smallest undesired eigenvalue of T. That is the eigenvalue closest to the desired eigenvalues.

Table 14. (Cont'd)

Name	Type	Dimension	Classification	Remarks
IERR	integer	scalar	0	An error completion code. Set to -2 if too many calls to OP and -3 if an error flag in an EISPACK subroutine is encountered.

7.2 INTERNAL VARIABLES

Table 15. Internal Variables

Name	Type	Remarks
ANORM	real	Current estimate of $\|\|A\|\|$. Computed as the infinity norm of the tridiagonal matrix obtained by taking the 2-norm of the blocks of T. This estimate is modified to reflect any user supplied eigenvalues.
BNORM	real	Holds the 2-norm (largest singular value) of the previous off diagonal block (see SINGL). Needed in the update formula for the T-recurrence.
ENOUGH	logical	Used to keep track of whether enough desirable eigenvalues have been found.
EPS	real	Set to an approximation to the relative machine precision by the repeated halving technique.
EPS1	real	Indicates the desired accuracy of the eigenvalues computed by TRIDIB. When set to 0.0 the eigenvalues are found to working accuracy.

Table 15. (Cont'd)

Name	Type	Remarks
EPSRT	real	Set to the square root of EPS. Used in assessing the orthogonality coefficients to determine if a given eigenvector is good. (The vector is good if $\omega_{ij} \leq \sqrt{\epsilon}\|A\|$).
I	integer	Used as a DO loop variable in many places.
I1	integer	Used as an array subscript for manipulating vectors in forming good Ritz vectors and starting vectors. Also used as a temporary location for swapping integers.
IER	integer	Used as the error completion code variable in all calls to EISPACK subroutines. If a nonzero completion code is encountered SNWLA is aborted with IERR = -3.
INDG	integer	Used as the index (in T) of the smallest eigenvalue of T which has not been declared good.
IURAND	integer	Used as the seed for the pseudo-random number generator URAND.
J	integer	Used as the current dimension of the matrix T. Thus J equals NBLOCK times the number of Lanczos steps taken.
K	integer	Used as the secondary DO loop index (after I).
L	integer	Used as the tertiary DO loop index (after K). Also used as the secondary array subscript or DO loop limit (after M).
LB	real	Needed for EISPACK subroutine TRIDIB. LB is never referenced.
M	integer	Used as the primary array subscript or DO loop limit.
NG	integer	Used to count the number of good eigenpairs found at any step. If this number is bigger than the number known previously (NGOOD) then all the good Ritz vectors are recomputed.

Table 15. (Cont'd)

Name	Type	Remarks
NGOOD	integer	The number of good Ritz pairs currently known. Just before a restart, NGOOD is equal to the number of new eigenpairs which will become permanent vectors when the algorithm starts over.
NLEFT	integer	The number of eigenvalues remaining to be found (= NVAL - NPERM).
NSTART	integer	Set to the number of unacceptable vectors. This number is then used to construct the appropriate linear combination of these vectors for the new starting block if needed.
NTHETA	integer	Set to the number of eigenvalues of T which are to be computed. This is usually NLEFT+1 but may not be larger than J/2 to prevent any confusion with converging eigenvalues at the large end of the spectrum. If NTHETA=NLEFT+1, DELTA is updated and then NTHETA is set to NLEFT.
NUMBER	integer	The actual number of vectors in VEC. (Equal to NPERM+NGOOD).
NV	integer	Set to MAXJ*(2*NBLOCK+1). Used as a parameter in calls to BANDV.
PNORM	real	Set to the largest eigenvalue (in absolute value) of the desired eigenvalues. Used instead of ANORM in evaluating the accuracy of the desired eigenvalues. Thus NFIG decimal digits of accuracy are obtained in the eigenvalues relative to themselves rather than to ANORM.
RNORM	real	Set to the largest eigenvalue (in absolute value) of the permanent eigenpairs. Used in updating PNORM.

Table 15. (Cont'd)

Name	Type	Remarks
SINGL	real	Set to the 2-norm (largest singular value) of BET. Passed on to BNORM. Used in the update formulas for the τ-recurrence.
SINGS	real	Set to the smallest singular value of BET. Used in the update formula for the τ-recurrence.
TEMP	real	Used as a temporary storage location.
TEST	logical	Used as a flag in three places. TEST is .TRUE. if it is necessary to reorthonormalize the current Lanczos block. TEST is .TRUE. if a restart is necessary. Later TEST is modified to indicate that a restart is necessary and starting vectors must be computed.
THETA	real	Computed as the eigenvalue of T of index NLEFT. If THETA is sufficiently accurate then all the desired eigenvalues are computed and examined.
THETG	real	Computed as the eigenvalue of T of index INDG. If THETG is found to be good all the desired eigenvalues are computed and examined.
TOLA	real	Set to UTOL*PNORM (or RNORM). Used as the acceptance tolerance. An eigenpair (y,θ) is acceptable if $\min(\rho_{ji}^2/(\delta-\theta),\beta_{ji}) \leqslant$ TOLA , where β_{ji} is the residual norm of the Ritz pair (y,θ).
TOLG	real	Set to EPSRT*ANORM. Used to determine if an eigenpair is good. An eigenpair is good if $\omega_{ji} \leqslant$ TOLG

Table 15. (Cont'd)

Name	Type	Remarks
		where ω_{ji} is the orthogonality coefficient of the eigenpair.
UB	real	Needed for the EISPACK subroutine TRIDIB. UB is never referenced.
UTOL	real	Set to max(N*EPS,10**(-NFIG)) used as the relative acceptability tolerance.

7.3 SEQUENCE BLOCKS

In reference to the listing of SNWLA given Appendix 2:

Table 16. Sequence Blocks

Sequence Numbers	Remarks
10-1080	Initialize declarations and comments.
1090-1170	Initialize IURAND.
1180-1290	Initialize EPS.
1300-1410	Initialize other parameters.
1420-1580	Replace zero vectors in starting block by random vectors.
1590-1650	Start the τ-recurrence, if necessary.
1660-1710	Reset the Lanczos parameters. TEST is set to .TRUE. to indicate that the starting block must be orthonormalized.
1720-3420	The Lanczos step.
1770	Update J.
1780-2010	Selective orthogonalization.

Table 16. (Cont'd)

Sequence Numbers	Remarks
2020-2250	Reorthonormalize the block and update BET.
2290	The call to OP.
2310-2320	The call to IOVECT.
2320-2540	Computation of P2 = P2 - PO*BET - P1*ALP.
2550-2680	Local reorthogonalization.
2690-2860	Store ALP and BET in T.
2870-3000	Negate T if needed.
3010-3070	Shift the blocks and orthonormalize the newest one.
3080-3280	Compute the 2-norm of ALP and the largest and smallest singular values of BET. Note that the EISPACK version of SVD does not order the singular values.
3290-3420	Update the T-recurrence if needed.
3430-3540	On the first two steps only, don't examine any eigenvalues of T.
3550-4210	Compute and examine 2 eigenvalues of T to see if all the desired eigenvalues should be examined.
3550-3660	Set some parameters.
3670-3740	Reduce T to tridiagonal form.
3750-3760	Check to see if a restart is needed.
3770-3870	Compute THETG and S the smallest nongood eigenpair of T and the corresponding resi al norm and orthogonality coefficient.
3880-4030	If NLEFT = 0 (check run) see if the check is successful. A check run is successful if at least 6 steps have been taken and the residual norm interval

Table 16. (Cont'd)

Sequence Numbers	Remarks
	around THETG does not extend to the left of the accuracy interval around the largest permanent value. That is the eigenvalue of A closest to THETG is greater than or equal to the largest permanent value to within the desired accuracy. On the other hand if THETG is smaller than the largest permanent value then the largest permanent value is thrown away, NLEFT is set to 1 and the Lanczos run continues as a "normal" run.
4040-4070	If NLEFT $\neq$ 0 then the orthogonality coefficient is examined to see if the eigenpair is now good. If it is than all the desired eigenpairs are examined.
4080-4210	THETA and S, the (NLEFT)$^{\underline{th}}$ smallest eigenpair is computed along with the corresponding residual norm and orthogonality coefficient. (Except that no examination is made if NLEFT $>$ J/2 since examining eigenvalues from the wrong end of the spectrum can lead to spurious results.) If this eigenvalue has changed from the corresponding eigenvalue computed at the previous step by less than one tenth of the desired accuracy, then all the desired eigenpairs are computed and examined.

Table 16. (Cont'd)

Sequence Numbers	Remarks
4220-4680	This section computes min(NLEFT+1,J/2) eigenvalues of T. If indicated NLEFT is updated. If NLEFT is increased some permanent vectors are discarded, all the good vectors are discarded (if any) and more eigenvalues are computed. If possible DELTA is updated. Then the corresponding residual norms and orthogonality coefficients are computed.
4690-5130	This section examines the computed eigenpairs, first to see whether all the computed eigenpairs are sufficiently accurate and then to see if more good eigenpairs have been found. In either case it is necessary to compute some Ritz vectors.
5140-5510	The previous section divided (using IND) the eigenvectors of T into two sets, namely those vectors corresponding to Ritz vectors needed in their own right and those needed only for constructing starting vectors (or not at all). This section sorts the eigenvectors of T so that the first set comes first.
5520-5800	If needed (TEST = .TRUE.) this section forms the appropriate linear combinations of the second set of vectors for forming starting vectors.
5810-5890	This stores the residual norms of the new permanent vectors (if any) in RES for use in the τ-recurrence.

Table 16 (Cont'd)

Sequence Numbers	Remarks
5900-6290	This forms the Ritz vectors (including the starting vectors) by sequentially recalling the Lanczos vectors.
6300-6430	This resets the τ-recurrence if the algorithm is not starting over.
6450-6720	This sorts the permanent vectors.
6730-6750	This updates NPERM, NLEFT, and RNORM.
6760-7150	This orthonormalizes the permanent vectors ordered by increasing residual norm.
7160-7330	This section decides whether to start over or whether to terminate.
7340-7480	This section sets RARITZ to .TRUE. if too large a cluster of eigenvalues was found.
7480	Normal exit.
7490-7640	This sets various error conditions before exiting.

8. SNPPLA

SNPPLA post processes the eigenpairs computed by SNWLA. If needed a final Rayleigh-Ritz procedure is performed on the eigenvectors. Then the Rayleigh quotients and residual norms are computed. Finally the accuracy estimates are computed.

8.1 THE CALLING SEQUENCE

SUBROUTINE SNPPLA(OP,IOVECT,N,NPERM,NOP,NMVAL,VAL,NMVEC,VEC,NBLOCK,H,P,
Q,DELTA,SMALL,RARITZ,IERR).

Table 17. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
OP	external	-	SI	Forms matrix vector products.
IOVECT	external	-	I	Needed only for Rayleigh-Ritz procedure.
N	integer	scalar	SI	Dimension of the matrix.
NPERM	integer	scalar	SI	The number of eigenvectors.
NOP	integer	scalar	SI-SO	The cumulative number of calls to OP.
NMVAL	integer	scalar	SI	The row dimension of VAL.
VAL	real	NMVAL*4	SO	The eigenvalues and accuracy estimates.
NMVEC	integer	scalar	SI	The row dimension of VEC.
VEC	real	NMVEC*NPERM	SI-O	The eigenvectors, which are modified only if a Rayleigh-Ritz procedure is computed.
NBLOCK	integer	scalar	SI	The blocksize. Used to minimize the number of calls to OP.
H	real	NPERM*NPERM	W	Holds the reduced matrix in the Rayleigh-Ritz procedure.
P	real	N*BLOCK	W	Used for forming block matrix vector products.
Q	real	N*BLOCK	W	Used for forming block matrix vector products.

Table 17. (Cont'd)

Name	Type	Dimension	Classification	Remarks
DELTA	real	scalar	SI-O	Input as the eigenvalue of A closest to the desired eigenvalues. If SMALL is .FALSE. DELTA must be negated to account for the fact that SNWLA was implicitly using -A as the matrix.
SMALL	logical	scalar	SI	SMALL is .TRUE. if the leftmost eigenvalues are desired.
RARITZ	logical	scalar	SI	RARITZ is .TRUE. if a Rayleigh-Ritz procedure is needed.
IERR	integer	scalar	I-O	The error indicator. Used here only to indicate a failure of an EISPACK subroutine (IMTQL2).

Note that the workspaces P and Q are needed to form block matrix vector products since the array VEC has row dimension NMVEC instead of N and the subroutine OP assumes a row dimension of N.

8.2 INTERNAL VARIABLES

Table 18. Internal Variables

Name	Type	Remarks
I	integer	Primary DO loop index.
IER	integer	Error completion code in a call to IMTQL2. A nonzero completion is almost impossible. If it occurs the code sets IERR to -3 and exits.

Table 18. (Cont'd)

Name	Type	Remarks
J	integer	Secondary DO loop index.
K	integer	Tertiary DO loop index.
L	integer	Array subscript.
M	integer	DO loop limit.
TEMP	real	Temporary storage. In the Rayleigh-Ritz procedure TEMP is used to negate the matrix H when SMALL = .FALSE. This forces IMTQL2 to return the eigenvectors sorted in the appropriate order.

8.3 SEQUENCE BLOCKS

In reference to the listing of SNPPLA given in Appendix 2:

Table 19. Sequence Blocks

Sequence Numbers	Remarks
10-200	Initial declarations and comments.
210-770	Construction of the reduced matrix H needed in the Rayleigh-Ritz procedure. The initial eigenvectors are stored by calls to IOVECT.
780-910	Spectral decomposition of H using EISPACK subroutines.
920-1280	Formation of the Ritz vectors (improved eigenvectors) as linear combinations of the original eigenvectors.
1290-1720	Computation of the Rayleigh quotient and residual norms of the eigenvectors.
1730-1870	Computation of the accuracy estimates and exit.

9. INTERVAL TYPE PROBLEMS

In this chapter we discuss the differences between SNLASO, SNWLA, and SNPPLA discussed earlier and SILASO, SIWLA, and SIPPLA which solve interval type problems.

9.1 PROBLEM DEFINITION

For interval type problems the user specifies XL and XR the left and right endpoints of the excluded interval and SILASO is supposed to find all the eigenvalues of the given matrix A outside the excluded interval. Thus it is necessary to examine eigenvalues at both ends of the spectrum of T. Furthermore problems occur if an eigenvalue of A lies pathologically close to one of the endpoints. For numerical reasons alone it is impossible to specify as sharp a boundary as the variables XL and XR suggest. Instead it is necessary to introduce a fuzzy region around each endpoint related to the desired accuracy in the eigenvalues. An eigenvalue which equals XL to the desired accuracy should be returned to the user even if computing it to working accuracy might disclose that it actually lies inside the excluded interval.

However while the eigenvector is returned to the user, the eigenvalue is explicitly set to the boundary so that all the returned eigenvalues do lie outside the excluded interval. Any such eigenpairs are marked to indicate that the returned eigenvalue is not the Rayleigh quotient of the eigenvector.

In the following sections we indicate the differences in the subroutines.

9.2 SILASO

9.2.1 The Calling Sequence

SUBROUTINE SILASO(OP,IOVECT,N,XL,XR,NFIG,NPERM,NMVAL,VAL,NMVEC,MAXVEC, VEC,NBLOCK,MAXOP,MAXJ,WORK,IND,IERR).

Table 20. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
~~NVAL~~				Not needed by SILASO.
XL	real	scalar	SI	Left endpoint.
XR	real	scalar	SI	Right endpoint.
MAXVEC	integer	scalar	SI	The column dimension of VEC. The maximum number of Ritz vectors which can be stored.

9.2.2 Internal Variables

Table 20. Internal Variables

Name	Type	Remarks
~~DELTA~~		Replaced by DELTAL and DELTAR.
DELTAL	real	Returned from SIWLA as the excluded eigenvalue closest to XL. Used in SIPPLA to compute the accuracy estimates.
DELTAR	real	Returned from SIWLA as the excluded eigenvalue closest to XR. Used in SIPPLA to compute the accuracy estimates.
NP	integer	Set to NPERM and passed to SIPPLA. NP is used to dimension arrays in SIPPLA. This legalizes changing the value of NPERM which may be necessary in SIPPLA.
~~NV~~		Not used.
~~SMALL~~		Not used.

9.2.3 Other Changes

In examining user supplied eigenpairs, SILASO will abort (IERR = -4) if a user supplied eigenvalue is inside the excluded interval.

9.3 SIWLA

There are two major differences between SIWLA and SNWLA. Since eigenvalues at both ends of the spectrum of A are of interest, SIWLA must examine both ends of the spectrum of T at each step. Also the termination criterion for SIWLA is rather different than for SNWLA. SIWLA must continue until the most extreme excluded eigenvalues have settled down enough to know that they are not going to drift into the desired region. Thus the first eigenvalues computed at each step are DELTAL and DELTAR, the most extreme excluded eigenvalues (of T). Only if their residual norm intervals do not overlap the boundary are the desired eigenvalues investigated to see if they are acceptable. Finally the most extreme non good eigenvalues are examined to see if a new good Ritz vector must be computed.

9.3.1 Calling Sequence

SUBROUTINE SIWLA(OP,IOVECT,N,XL,XR,NFIG,NPERM,VAL,NMVEC,MAXVEC,VEC,NBLOCK, MAXOP,MAXJ,NOP,PO,P1,P2,RES,TAU,OTAU,T,ALPHA,BETA,BETA2,ALP,BET,RV,RV6,S,IND, RARITZ,DELTAL,DELTAR,IERR).

Table 21. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
NVAL				Not needed.
MAXVEC	integer scalar		SI	The column dimension of VEC.
SMALL				Not needed.
DELTA				Replaced by DELTAL and DELTAR.

Table 21. (Cont'd)

Name	Type	Dimension	Classification	Remarks
DELTAL	real	scalar	SO	Smallest excluded eigenvalue.
DELTAR	real	scalar	SO	Largest excluded eigenvalue.

9.3.2 Internal Variables

Table 22. Internal Variables

Name	Type	Remarks
AXL	real	XL perturbed (to the right) by the desired accuracy. Thus any eigenvalue to the right of AXR is known to be not wanted.
AXR	real	XR perturbed (to the left) by the desired accuracy.
DONE	logical	A flag set to .TRUE. when DELTAL and DELTAR have settled down.
ENOUGH		Replaced by DONE.
INDAL	integer	The index of DELTAL in T.
INDAR	integer	The index of DELTAR in T.
INDG		Replaced by INDGL and INDGR.
INDGL	integer	The index of THETGL in T.
INDGR	integer	The index of THETGR in T.
NLEFT		Not needed.
NUML	integer	The number of eigenvalues of T less than AXL.
NUMR	integer	The number of eigenvalues of T greater than AXR.
PNORM		Not needed.
RNORM		Not needed.
THETA		Replaced by THETAL and THETAR.

Table 22. (Cont'd)

Name	Type	Remarks
THETAL	real	The largest desired eigenvalue of T less than AXL (of index INDAL-1).
THETAG	real	The smallest desired eigenvalue of T greater than AXR (of index INDAR+1).
THETC		Replaced by THETGL and THETGR.
THETGL	real	The smallest non good eigenvalue (of index INDGL in T).
THETGR	real	The largest non good eigenvalue (of index INDGR in T).
TOLA	real	Set to UTOL*max (XR, -XL) and used as the acceptability criterion.

9.3.3 Other Changes

The necessity of determining which side of the boundary an eigenvalue lies leads to two additional IERR codes. If J = MAXJ is reached when no desired eigenvalues are found but DELTAL or DELTAR has not settled down then IERR is set to -5. If NOP = MAXOP occurs in the same situation then IERR is set to -6. In either case the vectors corresponding to DELTAL and DELTAR are put in the starting block (as they are anytime the algorithm starts over) so that SILASO can be immediately recalled to continue working on the problem.

Furthermore, if more than MAXVEC eigenvalues are found it is necessary to stop. All acceptable eigenvectors are computed and the rest are put in the starting block. 100* (the number of surplus eigenvalues) is subtracted from IERR to indicate this result.

9.4 SIPPLA

The major difference between SIPPLA and SNPPLA is that SIPPLA must examine the eigenvalues to see if any of them lie inside the excluded interval. If an eigenvalue's residual norm interval ($[\theta_i - \beta_{ji}, \theta_i + \beta_{ji}]$) lies entirely inside the excluded interval, that eigenvalue is deleted (and the appropriate DELTA is redefined). If the residual norm interval overlaps the boundary then three changes are made. The eigenvalue is set equal to the boundary, the residual norm is recomputed and made negative, and 10 is subtracted from IERR. If ten or more such eigenvalues occur than IERR will be misleading since IERR ≤ -100 usually indicates that too many eigenvalues where found but this is quite unlikely.

9.4.1 Calling Sequence

SUBROUTINE SIPPLA(OP,IOVECT,N,XL,XR,NP,NPERM,NOP,NMVAL,VAL,NMVEC,VEC,NBLOCK
H,P,Q,DELTAL,DELTAR,RARITZ,IERR).

Table 23. The Calling Sequence

Name	Type	Dimension	Classification	Remarks
XL	real	scalar	SI	Left endpoint.
XR	real	scalar	SI	Right endpoint.
NP	integer	scalar	SI	Equal to NPERM. Used to dimension arrays.
NPERM	integer	scalar	SI-O	May change value.
~~DELTA~~				Replaced by DELTAL and DELTAR.
DELTAL	real	scalar	SI-O	Smallest excluded eigenvalue.

Table 23. (Cont'd)

Name	Type	Dimension	Classification	Remarks
DELTAR	real	scalar	SI-O	Largest excluded eigenvalue.
~~SMALL~~				Not needed.

9.4.2 Internal Variables

No changes.

REFERENCES

[1] J. Cullum and W. E. Donath, "A Block Generalization of the Symmetric S-Step Lanczos Algorithm," Report #RC 4845 (#21570), IBM Thomas J. Watson Research Center, Yorktown Heights, New York, (1974).

[2] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack Users' Guide*, SIAM, 1979.

[3] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Series in Automatic Computing, Prentice-Hall, 1977.

[4] B. S. Garbow, J. M. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines - Eispack Guide Extensions*, Lecture Notes in Computer Science 51, Springer-Verlag, 1977.

[5] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for FORTRAN Usage," to appear in *TOMS*.

[6] J. Lewis, "Algorithms for Sparse Matrix Eigenvalue Problems," Technical Report STAN-CS-77-595, Computer Science Department, Stanford University (1977).

[7] C. C. Paige, "The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices," Ph.D. Thesis, University of London (1971).

[8] C. C. Paige, "Computational Variants of the Lanczos Method for the Eigenproblem," *J. Inst. Math. Applics.* 10, 373-81, 1972.

[9] B. N. Parlett and D. S. Scott, "The Lanczos Algorithm with Selective Orthogonalization," *Math. of Comp.* 33, 217-38, 1979.

[10] A. Ruhe, "Implementation Aspects of Band Lanczos Algorithms for Computation of Eigenvalues of Large Sparse Symmetric Matrices," *Math.Comp.* 33, 680-7, 1979.

[11] B. G. Ryder and A. D. Hall, "PFORT Verifier," Computing Science Technical Report #12, Bell Labs, Murray-Hill, NJ, (1975).

[12] D. S. Scott, "Analysis of the Symmetric Lanczos Process," Ph.D. Thesis, ERL Technical Report # M78/40, June 1978, Berkeley, CA 94720.

[13] B. T. Smith et al., *Matrix Eigensystem Routines - Eispack Guide*, Lecture Notes in Computer Science 6, 2nd edition, Springer-Verlag, 1976.

[14] R. Underwood, "An Iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems," Ph.D. Thesis, Stanford University, STAN-CS-75-496 (1975).

APPENDIX 1

User guides for SNLASO/DNLASO

and SILASO/DILASO

SNLASO/DNLASO

A FORTRAN IV subroutine for determining a few eigenvalues and eigenvectors at one end of the spectrum of a large sparse symmetric matrix. SNLASO is in single precision and DNLASO is in double precision. This documentation explicitly describes SNLASO. For DNLASO all subroutine and function names start with D instead of S (except SVD) and all floating point variables are double precision.

David S. Scott
Union Carbide Corporation, Nuclear Division
Oak Ridge, TN 37830
July, 1979

1. Purpose

The FORTRAN IV subroutine SNLASO determines a few eigenvalues and eigenvectors at one end of the spectrum of a large sparse symmetric matrix, hereafter called A. SNLASO uses the block Lanczos algorithm with selective orthogonalization to compute Rayleigh-Ritz approximations to eigenpairs of A.

2. Usage

A. Calling sequence.

The subroutine statement is

```
SUBROUTINE SNLASO (OP, IOVECT, N, NVAL, NFIG, NPERM,
   NMVAL, VAL, NMVEC, VEC, NBLOCK, MAXOP, MAXJ, WORK, IND,
   IERR)
```

On input:

OP	specifies a user supplied subroutine for entering information about the matrix A with calling sequence OP(N,M,P,Q). See section B. for further information.
IOVECT	specifies a user supplied subroutine for storing and recalling vectors with calling sequence IOVECT (N,M,Q,J,K). See section B. for further information.
N	specifies the dimension of the matrix.

NVAL — specifies which eigenvalues are desired. abs (NVAL) eigenvalues are to be found. If NVAL < 0 the algebraically smallest (leftmost) are found while if NVAL > 0 the algebraically largest (rightmost) are found. NVAL must not be zero.

NFIG — specifies the number of decimal digits of accuracy desired in the eigenvalues.

NPERM — is an integer variable specifying the number of eigenpairs presupplied by the user. In most cases NPERM will be zero. See section H. for information on using NPERM > 0. NPERM must not be less than zero.

NMVAL — specifies the row dimension of the real array VAL. NMVAL must be greater than or equal to abs (NVAL).

VAL — is a two dimensional real array with NMVAL rows and at least four columns. If NPERM > 0 on input, VAL must contain certain information. See section H. for details.

NMVEC — specifies the row dimension of the real array VEC. NMVEC must be greater than or equal to N.

VEC — is a two dimensional real array with NMVEC rows and at least abs (NVAL) columns. If NPERM > 0 on input VEC must contain certain information. See section H. for details.

NBLOCK — specifies the number of vectors in each Lanczos block. See section F. for guidelines in choosing a value for NBLOCK. NBLOCK must be greater than zero and less than or equal to MAXJ/6.

MAXOP — specifies an upper bound on the number of calls to the subroutine OP. SNLASO terminates when this maximum is reached. See section G. for guidelines in choosing a value for MAXOP.

MAXJ — specifies an indication of the available storage, see WORK in this section and IOVECT in section B. The larger the value of MAXJ the faster the convergence rate of the algorithm. However, there is no advantage in having MAXJ > MAXOP*NBLOCK. MAXJ must not be less than 6* NBLOCK.

WORK — is a one dimensional real array at least as large as

NBLOCK*(3*N + 2*NBLOCK) + MAXJ* (3*NBLOCK + abs(NVAL) + 6) + 3*abs(NVAL)

used for workspace. The first N*BLOCK elements of work must contain the starting vectors for the algorithm. See section E. for details.

IND — is an integer array of dimension at least

abs (NVAL),

used for workspace.

IERR — is an integer variable.

On output:

NPERM — is the number of eigenpairs now known.

VAL — contains information about the eigenpairs. The first column of VAL contains the eigenvalues, ordered from the most extreme one inward. The second, third, and fourth columns of VAL contain information on the accuracy of the eigenvalues and eigenvectors. See section D. for details.

VEC — contains the corresponding eigenvectors.

WORK — if IERR $\neq$ 0, the first N*BLOCK elements of WORK will contain vectors for restarting the algorithm. See section E. for details.

IND (1) — contains the actual number of calls to the subroutine OP. In some circumstances this may be slightly larger than MAXOP.

IERR — is an error completion code. The normal completion code is zero. See section C. for the interpretation of non-zero completion codes.

B. User supplied subroutines.

The two user supplied subroutines must be declared EXTERNAL in the calling program and must conform as follows:

OP (N,M,P,Q). P and Q are N x M real arrays. Q should be returned as AP where A represents the matrix whose eigenpairs are to be determined.

M will never be larger than NBLOCK but it may be smaller. This subroutine is the only way in which the matrix enters the calculation, so the user is free to take advantage of any sparsity structure in the matrix. The user should adequately test the subroutine OP because SNLASO has no way of detecting errors made in OP.

IOVECT (N,M,Q,J,K). Q is an N x M real array. M will never be larger than NBLOCK but it may be smaller. IOVECT is used to store Lanczos vectors as they are computed and to periodically recall all the currently stored Lanczos vectors. If K = 0 then the M columns of Q should be stored as the (J - M + 1) th through the J th Lanczos vectors. If K = 1 then the columns of Q should be returned as the (J - M + 1) th through the J th Lanczos vectors which were previously stored.

The Lanczos vectors are computed sequentially. They are stored by calls to IOVECT with K = 0 and increasing values of J up to some internally derived value J = I which signals a pause. These vectors are then recalled by calls to IOVECT with K = 1 and the same sequence of J values. The first J value of any sequence is equal to M. After the pause more Lanczos vectors are computed and these are stored by calls to IOVECT with K = 0 and J values greater than I until the next pause at which time all the Lanczos vectors currently stored are recalled with calls to IOVECT with K = 1 and J = M, ...

After any pause the algorithm may discard the current Lanczos vectors and start a new sequence of Lanczos vectors by a call to IOVECT with K = 0 and J = M. At subsequent pauses only the current sequence of Lanczos vectors is recalled. In solving a problem SNLASO may pause many times and discard the previous Lanczos vectors several times before converging to the final solution.

The largest value to J which can appear in a call to IOVECT is J = MAXJ.

We give two examples for IOVECT. The first example requires that logical unit 20 be assigned to a secondary storage medium.

```
SUBROUTINE IOVECT (N,M,Q,J,K)
INTEGER N,M,J,K,I,L
DIMENSION Q(N,M)
IF (J.EQ.M) Rewind 20
IF (K.EQ.0) Write (20) ((Q(I,L), I = 1,N), L = 1,M)
IF (K.EQ.1) Read (20) ((Q(I,L), I = 1,N), L = 1,M)
RETURN
END
```

The Lanczos vectors can also be kept in fast store. In this example we assume that N $\leq$ 100 and MAXJ $\leq$ 50.

```
      SUBROUTINE IOVECT (N,M,Q,J,K)
      INTEGER N,M,J,K,I,L,L1
      DIMENSION Q(N,M)
      COMMON QVEC (100,50)
      IF (K. EQ. 1) GO TO 30
      DO 20 L = 1,M
        L1 = J - M + L
        DO 10 I = 1,N
           QVEC (I,L1) = Q(I,L)
10    CONTINUE
20    CONTINUE
      RETURN
30    DO 50 L = 1, M
        L1 = J - M + L
        DO 40 I = 1, N
        Q (I,L) = QVEC (I, L1)
40    CONTINUE
50    CONTINUE
      RETURN
      END
```

C. ERROR completion codes.

IERR = 0 indicates a normal completion. abs (NVAL) eigenpairs have been determined. See section D. for the information returned.

IERR > 0 and IERR < 1024 indicates that some inconsistency in the calling parameters was discovered and no computation was performed.

```
    1-bit is set if N < 6*NBLOCK
    2-bit is set if NFIG ≤ 0
    4-bit is set if NMVEC < N
    8-bit is set if NPERM < 0
   16-bit is set if MAXJ < 6*NBLOCK
   32-bit is set if abs (NVAL) < max (1,NPERM)
   64-bit is set if abs(NVAL) > NMVAL
  128-bit is set if abs(NVAL) > MAXOP
  256-bit is set if abs(NVAL) > MAXJ/2
  512-bit is set if NBLOCK < 1
```

Thus IERR can be decoded to determine the errors. For example, IERR = 68 indicates both NMVEC < N and abs(NVAL) > NMVAL. IERR may take on any value between 1 and 1023 indicating all combinations of the above conditions.

IERR = -1 can occur only if NPERM > 0 on input. It indicates that either some user supplied eigenvector was zero or that significant cancellation occured when the user supplied vectors were orthogonalized. Some modification of the user supplied eigenvectors will have occurred but no other computation will have been done.

IERR = -2 indicates that MAXOP calls to the subroutine OP occurred without finding the desired eigenvalues. Partial information is returned in this case, see section G. When IERR = -2, the first N*NBLOCK elements of work contain the best vectors for restarting the algorithm. Thus SNLASO may be immediately recalled to continue working on the problem

IERR = -3 indicates a non-zero error completion code was encountered after a call to an EISPACK subroutine. EISPACK is a certified subroutine package. Errors are due to improper inputs. The following is a list of possible causes for an IERR = -3 completion:

1. Improper calling sequence for SNLASO.
2. Insufficient storage in the array WORK.
3. Mixture of single and double precision.
4. Improper version of EISPACK for the machine used.

IERR = -8 indicates that disastrous loss of orthogonality occurred. Usually due to errors in the user supplied subroutines OP or IOVECT.

D. Information returned when IERR = 0.

IERR = 0 indicates that the desired eigenpairs have been found. The eigenvalues are in the first column of VAL. If NVAL < 0 the eigenvalues are in ascending order (smallest at the top) while if NVAL > 0 the eigenvalues are in descending order. The corresponding orthonormal eigenvectors are in the first abs(NVAL) columns of VEC. The second column of VAL contains the residual norms $\rho_i (=\|Ay_i - y_i\theta_i\|$ for the eigenvalue θ_i and its associated eigenvector y_i) which are bounds on the accuracy of the eigenvalues.

In most cases the residual norm is a gross underbound on the accuracy of an eigenvalue. To obtain a more realistic estimate, the program remembers δ, its best estimate of the eigenvalue of the matrix which is closest to the desired eigenvalues. The third column of VAL is set to $\rho_i^2/\text{abs}(\theta_i - \delta)$ which is a much more realistic estimate of accuracy of the eigenvalues. The fourth column of VAL contains $\rho_i/\text{abs}(\theta_i - \delta)$ which estimates the accuracy of the eigenvectors.

If the user has supplied some eigenpairs of the matrix, it is possible that some of these eigenpairs have been discarded in favor of eigenpairs computed by the algorithm. (See section H. for additional information.)

E. Choosing the starting vectors.

SLASO requires NBLOCK starting vectors to be stored in the first N*NBLOCK elements of the array WORK. Zero vectors are replaced by randomly chosen vectors so that a set of random starting vectors may be selected by simply initializing the first N*NBLOCK elements of WORK to zero. However, convergence is enhanced if the starting vectors are chosen to have large components in the directions of the desired eigenvectors. Therefore, if the user knows approximations to the desired eigenvectors he should choose his starting vectors as linear combinations of these approximations.

If some of the desired eigenpairs are already known to sufficient accuracy, it is possible to avoid having SNLASO recompute these eigenpairs. See section H. for details.

F. Choosing a value for NBLOCK.

NBLOCK specifies the number of vectors in each block of Lanczos vectors. Two factors may favor a large value for NBLOCK. The convergence of the algorithm is faster if NBLOCK is larger than the largest multiplicity of eigenvalues among the desired eigenvalues. For instance if a desired eigenvalue has multiplicity two, then NBLOCK equal to three or more is best. Even more important in some cases, if the matrix is stored on disk and brought in a slice at a time to form the matrix vector product then a large value of NBLOCK will lower the number of calls to OP and hence the number of disk accesses. On the other hand the number of vector inner products needed for each Lanczos step is a quadratic function of NBLOCK. Furthermore, the convergence of the algorithm is degraded if NBLOCK > $\sqrt{\text{MAXJ}}$. In conclusion if the matrix multiply is inexpensive a small value of NBLOCK (2 or 3) is best while if the matrix multiply is expensive larger values of NBLOCK are to be preferred. NBLOCK = 1 is recommended only if abs(NVAL) = 1 as well.

G. Choosing a value for MAXOP

SNLASO is an iterative procedure. The user may limit the effort by SNLASO on a given problem by choosing a value for MAXOP. If more than MAXOP calls to the subroutine OP are needed to solve the given problem, then SNLASO will terminate at that point and set IERR = -2.

If cost is not a factor and the subroutine OP is known to be reliable MAXOP should be set to N/NBLOCK. Choosing MAXOP much less than abs(NVAL) $\sqrt{\text{N/NBLOCK}}$ and repeatedly recalling SNLASO will delay convergence of the algorithm. Setting MAXOP < abs(NVAL) is not allowed while setting MAXOP < MAXJ/NBLOCK will waste the storage indicated by MAXJ.

H. Setting NPERM > 0.

SNLASO allows known eigenpairs to be input directly so that they need not be recomputed. The first column of VAL must contain the eigenvalues (in any order) and the second column of VAL must contain the residual norms ($\| Ay_i - y_i\theta_i \|$, for the eigenpair $\theta_i y_i$). The correct order of magnitude is sufficient. Columns 3 and 4 of VAL are arbitrary. The first NPERM columns of VEC must contain the eigenvectors (which will be orthonormalized by SNLASO). The eigenvectors associated with VAL(I,1) must be in the I th column of VEC.

The user supplied eigenvalues are counted toward the number of desired eigenpairs and so NPERM must be less than or equal to abs(NVAL). If in the course of the computation it appears that a user supplied eigenpair is not one of the desired eigenpairs, it will be discarded. In particular if NPERM = abs(NVAL), the algorithm will either confirm that the supplied eigenpairs are indeed the desired eigenpairs or it will discard one or more in favor of newly computed eigenpairs.

3. Applicability and Restrictions

SNLASO is designed to find a few extreme eigenpairs of a large sparse symmetric matrix. For small dense matrices the subroutines provided in EISPACK are to be preferred. It is not necessary for the matrix to be explicitly represented. It is only necessary to provide a subroutine OP to compute matrix-vector products. For example, consider the generalized eigenvalue problem $(A - \lambda M)x = 0$ where M is positive definite and can be factored as LL^T. The matrix $L^{-1}AL^{-T}$ can be implicitly coded in OP as a triangular solve, a matrix multiply, and another triangular solve. Thus a generalized eigenproblem can be reduced to a standard eigenproblem without the cost of explicitly forming $L^{-1}AL^{-T}$. More complex operators can also be handled efficiently.

SNLASO calls a number of subsidiary functions and subroutines, namely:

SNWLA which implements the block Lanczos algorithm with selective orthogonalization.

SNPPLA which post processes the output of SNWLA.

SMVPC which computes residual norms and orthogonality coefficients.

SORTQR which orthonormalizes a block of vectors.

SVZERO which zeroes a given vector.

URAND, a FORTRAN IV random number generator given in Forsythe, Malcolm, and Moler [6].

BANDR, BANDV, IMTQL1, IMTQL2, SVD, TRED1, TRED2, and TRIDIB, which are EISPACK Subroutines ([3], [4]).

SAXPY, SCOPY, SDOT, SNRM2, SSCAL and SSWAP which are subset of the BLAS (Basic Linear Algebra Subprograms) written by Lawson, et. al [7] and modified by Dongarra, et. al [8] for use in LINPACK. If the BLAS are available in assembly language they should be used in place of the FORTRAN IV source code provided.

The user must not use any of the above names in his driver program.

4. Discussion of method and algorithm.

The Lanczos algorithm is an efficient scheme for computing a series of vectors q_1, q_2, ... q_j which form an orthonormal basis for the Krylov subspace, span $(q_1, Aq_1, \ldots, A^{j-1}q_1)$.

At each step of the algorithm the Krylov subspace grows larger and one more Lanczos vector is added to the list. The Lanczos algorithm can be interrupted at any step and the Rayleigh-Ritz approximations to eigenpairs of A can be derived from the Krylov subspace quite easily. Thus the algorithm need only continue until the desired approximations a sufficiently accurate.

The block Lanczos algorithm (as described in detail by Underwood [5]) replaces each vector in the simple Lanczos algorithm by an orthonormal block of vectors. Block Lanczos has theoretical advantages over simple Lanczos with respect to finding multiple eigenvalues and has advantages in efficiency if the cost of forming a matrix-vector product is high.

Unfortunately finite precision arithmetic causes the vectors computed by the Lanczos algorithm (both simple and block) to lose orthogonality and approach linear dependence. To maintain robust independence among the Lanczos vectors, SNLASO augments the algorithm with selective orthogonalization which causes some of the Lanczos vectors to be orthogonalized against a few selected Ritz vectors, as described in [1] and [2].

The algorithm is terminated when the desired Ritz values are sufficiently accurate. If necessary, SNLASO then makes another Lanczos run to test for undisclosed multiplicities. Finally in some cases, SNLASO performs a Rayleigh-Ritz procedure on the determined eigenvalues to resolve any clusters.

5. References

[1] B. N. Parlett and D. S. Scott, "The Lanczos Algorithm with Selective Orthogonalization," *Math. of Comp.* 33, 217-238, 1979.

[2] D. S. Scott, "Analysis of the Symmetric Lanczos Process," Ph.D. Thesis, ERL technical report No. M78/40, June 1978, Electronics Research Lab, University of California, Berkeley, CA 94720

[3] B. T. Smith et al., *Matrix Eigensystem Routines - Eispack Guide*, Lecture Notes in Computer Science 6, 2nd edition, Springer-Verlag, 1976.

[4] B. S. Garbow, J. M. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines - Eispack Guide Extensions*, Lecture Notes in Computer Science 51, Springer-Verlag, 1977.

[5] R. Underwood, "An Iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems," Ph. D. Thesis, Stanford University, STAN-CS-75-496 (1975).

[6] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Series in Automatic Computing, Prentice-Hall.

[7] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for FORTRAN Usage," to appear in *TOMS*.

[8] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack Users' Guide*, SIAM, 1979.

SILASO/DILASO

A FORTRAN IV subroutine for determining all the eigenvalues and eigenvectors of a large sparse symmetric matrix outside a user defined excluded interval. SILASO is in single precision and DILASO is in double precision. This documentation explicitly describes SILASO. For DILASO all subroutine and function names start with D instead of S (except SVD) and all floating point variables are double precision.

David S. Scott
Union Carbide Corporation, Nuclear Division
Oak Ridge, TN 37830
July, 1979

1. Purpose

The FORTRAN IV subroutine SILASO determines all the eigenvalues and eigenvectors of a large sparse symmetric matrix, hereafter called A, outside a user defined excluded interval. SILASO uses the block Lanczos algorithm with selective orthogonalization to compute Rayleigh-Ritz approximations to the eigenpairs of A.

2. Usage

A. Calling sequence.

The subroutine statement is

```
SUBROUTINE SILASO (OP, IOVECT, N, XL, XR, NFIG, NPERM,
     NMVAL, VAL, NMVEC, MAXVEC, VEC, NBLOCK, MAXOP, MAXJ,
     WORK, IND, IERR)
```

On input:

OP	specifies a user supplied subroutine for entering information about the matrix A with calling sequence OP(N,M,P,Q). See section B. for further information.
IOVECT	specifies a user supplied subroutine for storing and recalling vectors with calling sequence IOVECT (N,M,Q,J,K). See section B. for further information.
N	specifies the dimension of the matrix.
XL	specifies the left endpoint of the excluded interval.

XR	specifies the right endpoint of the excluded interval.
NFIG	specifies the number of decimal digits of accuracy desired in the eigenvalues.
NPERM	is an integer variable specifying the number of eigenpairs presupplied by the user. In most cases NPERM will be zero. See section H. for information on using NPERM > 0. NPERM must not be less than zero.
NMVAL	specifies the row dimension of the real array VAL. NMVAL must be greater than or equal to MAXVEC.
VAL	is a two dimensional real array with NMVAL rows and at least four columns. If NPERM > 0 on input, VAL must contain certain information. See section H. for details.
NMVEC	specifies the row dimension of the real array VEC. NMVEC must be greater than or equal to N.
MAXVEC	specifies the maximum number of eigenpairs which can be determined. MAXVEC must not exceed the column dimension of the array VEC.
VEC	is a two dimensional real array with NMVEC rows and at least MAXVEC columns. If NPERM > 0 on input VEC must contain certain information. See section H. for details.
NBLOCK	specifies the number of vectors in each Lanczos block. See section F. for guidelines in choosing a value for NBLOCK. NBLOCK must be greater than zero and less than or equal to MAXJ/6.
MAXOP	specifies an upper bound on the number of calls to the subroutine OP. SILASO terminates when this maximum is reached. See section G. for guidelines in choosing a value for MAXOP.

MAXJ — specifies an indication of the available storage, see WORK in this section and IOVECT in section B. The larger the value of MAXJ the faster the convergence rate of the algorithm. However, there is no advantage in having MAXJ > MAXOP*NBLOCK. MAXJ must not be less than 6* NBLOCK.

WORK — is a one dimensional real array at least as large as

NBLOCK*(3*N + 2*NBLOCK) + MAXJ* (3*NBLOCK + MAXVEC + 6) + 3*MAXVEC)

used for workspace. The first N*BLOCK elements of work must contain the starting vectors for the algorithm. See section E. for details.

IND — is an integer array of dimension at least

MAXVEC ,

used for workspace.

IERR — is an integer variable.

On output:

NPERM — is the number of eigenpairs now known.

VAL — contains information about the eigenpairs. The first column of VAL contains the eigenvalues, ordered from the leftmost to the right. The second, third, and fourth columns of VAL contain information on the accuracy of the eigenvalues and eigenvectors. See section D. for details.

VEC — contains the corresponding eigenvectors.

WORK — if IERR $\neq$ 0, the first N*BLOCK elements of WORK will contain vectors for restarting the algorithm. See section E. for details.

IND (1) — contains the actual number of calls to the subroutine OP. In some circumstances this may be slightly larger than MAXOP.

IERR — is an error completion code. The normal completion code is zero. See section C. for the interpretation of non-zero completion codes.

B. User supplied subroutines.

The two user supplied subroutines must be declared EXTERNAL in the calling program and must conform as follows:

OP (N,M,P,Q). P and Q are N x M real arrays. Q should be returned as AP where A represents the matrix whose eigenpairs are to be determined. M will never be larger than NBLOCK but it may be smaller. This subroutine is the only way in which the matrix enters the calculation, so the user is free to take advantage of any sparsity structure in the matrix. The user should adequately test the subroutine OP because SILASO has no way of detecting errors made in OP.

IOVECT (N,M,Q,J,K). Q is an N x M real array. M will never be larger than NBLOCK but it may be smaller. IOVECT is used to store Lanczos vectors as they are computed and to periodically recall all the currently stored Lanczos vectors. If K = 0 then the M columns of Q should be stored as the (J - M + 1) th through the J th Lanczos vectors which were previously stored.

The Lanczos vectors are computed sequentially. They are stored by calls to IOVECT with K = 0 and increasing values of J up to some internally derived value J = I which signals a pause. These vectors are then recalled by calls to IOVECT with K = 1 and the same sequence of J values. The first J value of any sequence is equal to M. After the pause more Lanczos vectors are computed and these are stored by calls to IOVECT with K = 0 and J values greater than I until the next pause at which time all the Lanczos vectors currently stored are recalled with calls to IOVECT with K = 1 and J = M, ...

After any pause the algorithm may discard the current Lanczos vectors and start a new sequence of Lanczos vectors by a call to IOVECT with K = 0 and J = M. At subsequent pauses only the current sequence of Lanczos vectors is recalled. In solving a problem SILASO may pause many times and discard the previous Lanczos vectors several times before convergint to the final solution.

The largest value to J which can appear in a call to IOVECT is J = MAXJ.

We give two examples for IOVECT. The first example requires that logical unit 20 be assigned to a secondary storage medium.

```
      SUBROUTINE IOVECT (N,M,Q,J,K)
      INTEGER N,M,J,K,I,L
      DIMENSION Q(N,M)
      IF (J.EQ.M) REWIND 20
      IF (K.EQ.0) WRITE (20) ((Q(I,L), I = 1,N), L = 1,M)
      IF (K.EQ.1) READ (20) ((Q(I,L), I = 1,N), L = 1,M)
      RETURN
      END
```

The Lanczos vectors can also be kept in fast store. In this example we assume that N $\leq$ 100 and MAXJ $\leq$ 50.

```
      SUBROUTINE IOVECT (N,M,Q,J,K)
      INTEGER N,M,J,K,I,L,L1
      DIMENSION Q(N,M)
      COMMON QVEC (100,50)
      IF (K.EQ.1) GO TO 30
      DO 20 L = 1,M
      L1 = J - M + L
      DO 10 I = 1,N
      QVEC (I,L1) = Q(I,L)
   10 CONTINUE
   20 CONTINUE
      RETURN
   30 DO 50 L = 1,M
      L1 = J - M + L
      DO 40 I = 1,N
      Q(I,L) = QVEC (I,L1)
   40 CONTINUE
   50 CONTINUE
      RETURN
      END
```

C. Error completion codes.

IERR = 0 indicates a normal completion. NPERM eigenpairs have been determined. See section D. for the information returned.

IERR $>$ 0 and IERR $<$ 1024 indicates that some inconsistency in the calling parameters was discovered and no computation was performed.

```
    1-bit is set if N < 6* NBLOCK
    2-bit is set if NFIG ≤ 0
    4-bit is set if NMVEC < N
    8-bit is set if NPERM < 0
   16-bit is set if MAXJ < 6*NBLOCK
   32-bit is set if MAXVEC ≤ NPERM
   64-bit is set if MAXVEC > NMVAL
  128-bit is set if MAXVEC > MAXOP
  256-bit is set if XL > XR
  512-bit is set if NBLOCK < 1
```

Thus IERR can be decoded to determine the errors. For example, IERR = 68 indicates both NMVEC < N and MAXVEC > NMVAL. IERR may take on any value between 1 and 1023 indicating all combinations of the above conditions.

IERR = -1 can occur only if NPERM > 0 on input. It indicates that either some user supplied eigenvector was zero or that significant cancellation occured when the user supplied vectors were orthogonalized. Some modification of the user supplied eigenvectors may have occurred but no other computation will have been done.

IERR = -2 indicates that MAXOP calls to the subroutine OP occurred without finding the desired eigenvalues. Partial information is returned in this case, see section G. When IERR = -2, the first N*BLOCK elements of work contain the best vectors for restarting the algorithm. Thus SILASO may be immediately recalled to continue working on the problem.

IERR = -3 indicates a non-zero error completion code was encountered after a call to an EISPACK subroutine. EISPACK is a certified subroutine package. Errors are due to improper inputs. The following is a list of possible causes for an IERR = -3 completion:

1. Improper calling sequence for SILASO.
2. Insufficient storage in the array WORK.
3. Mixture of single and double precision.
4. Improper version of EISPACK for the machine used.

IERR = -4 can occur only if NPERM > 0 on input. It indicates that a user supplied eigenvalue lies inside the excluded interval. Some modification of the user supplied vectors may have occurred but no other computation will have been done.

IERR = -5 and IERR = -6 indicate that the program termiated without full assurance that all the desired eigenvalues had been located due to an eigenvalue near the boundary of the excluded interval which had not converged to sufficient accuracy. IERR = -5 if J = MAXJ while IERR = -6 if MAXOP calls to the subroutine OP occurred. To obtain further assurance that all the eigenvalues have been found it is possible to recall SILASO to continue working on the problem.

IERR = -7 indicates that more than 4*MAXVEC eigenvalues are found. The program terminates without computing any new eigenpairs.

IERR = -8 indicates that disastrous loss of orthogonality occurred. Usually due to errors in the user supplied subroutines OP or IOVECT.

IERR $\leq$ -10 indicates that some of the eigenvalues found by the program lie inside the excluded interval but have error bounds which overlap the boundary. Such eigenvalues are explicitly set equal to the boundary and marked as described in section D. The ten's digit of IERR indicates the number of such eigenvalues while the units digit indicates the same result as single digit IERR codes described above.

IERR $\leq$ - 100 indicates that more than MAXVEC (but not more than 4*MAXVEC) eigenvalues were found. The hundreds digit of IERR gives the number of extra eigenvalues found. Any eigenpairs returned by the program are correct and after raising the value of MAXVEC it is possible to immediately recall SILASO to keep working on the problem. Of course the extra storage space indicated by the larger value of MAXVEC must be available. The tens and units digits of the IERR code are as described above.

D. Information returned when IERR = 0.

IERR = 0 indicates that NPERM desired eigenpairs have been found. The eigenvalues are in the first column of VAL. The eigenvalues are in ascending order (smallest at the top). The corresponding orthonormal eigenvectors are in the first NPERM columns of VEC. The second column of VAL contains the residual norms $\rho_i (= \|Ay_i - y_i\theta_i\|$ for the eigenvalue θ_i and its associated eigenvector y_i) which are bounds on the accuracy of the eigenvalues.

In most cases the residual norm is a gross underbound on the accuracy of an eigenvalue. To obtain a more realistic estimate, the program remembers δ_L and δ_R the leftmost and rightmost excluded eigenvalue of A. The third column of VAL is set to $\rho_i^2/\text{MIN}(\theta_i - \delta_L, \delta_R - \theta_i)$ which is a much more realistic estimate of the accuracy of the eigenvalues. The fourth column of VAL contains $\rho_i/\text{MIN}(\theta_i - \delta_L, \delta_R - \theta_i)$ which estimates the accuracy of the eigenvectors.

If IERR $\leq$ 10, then the eigenvalues which have been moved are marked by setting the residual norm negative.

E. Choosing the starting vectors.

SILASO requires NBLOCK starting vectors to be stored in the first N*BLOCK elements of the array WORK. Zero vectors are replaced by randomly chosen vectors so that a set of random

starting vectors may be selected by simply initializing the first N*BLOCK elements of WORK to zero. However, convergence is enhanced if the starting vectors are chosen to have large components in the directions of the desired eigenvectors. Therefore, if the user knows approximations to the desired eigenvectors he should choose his starting vectors as linear combinations of these approximations.

If some of the desired eigenpairs are already known to sufficient accuracy, it is possible to avoid having SILASO recompute these eigenpairs. See section H. for details.

F. Choosing a value for NBLOCK.

NBLOCK specifies the number of vectors in each block of Lanczos vectors. Two factors may favor a large value for NBLOCK. The convergence of the algorithm is faster if NBLOCK is larger than the largest multiplicity of eigenvalues among the desired eigenvalues. For instance if a desired eigenvalue has multiplicity two, then NBLOCK equal to three or more is best. Even more important in some cases, if the matrix is stored on disk and brought in a slice at a time to form the matrix vector product then a large value of NBLOCK will lower the number of calls to OP and hence the number of disk accesses. On the other hand, the number of vector inner products needed for each Lanczos step is a quadratic function of NBLOCK. Furthermore, the convergence of the algorithm is degraded if NBLOCK $> \sqrt{MAXJ}$. In conclusion if the matrix multiply is inexpensive a small value of NBLOCK (2 or 3) is best, while if the matrix multiply is expensive larger values of NBLOCK are to be preferred. NBLOCK = 1 is not recommended unless required by storage limitations.

G. Choosing a value for MAXOP.

SILASO is an iterative procedure. The user may limit the effort by SILASO on a given problem by choosing a value for MAXOP. If more than MAXOP calls to the subroutine OP are needed to solve the given problem, then SILASO will terminate at that point and set IERR = -2.

If cost is not a factor and the subroutine OP is known to be reliable MAXOP should be set to N/BLOCK. Choosing MAXOP much less than MAXVEC $\sqrt{N/NBLOCK}$ and repeatedly recalling SILASO will delay convergence of the algorithm. Setting MAXOP $<$ MAXVEC is not allowed while setting MAXOP $<$ MAXJ/NBLOCK will waste the storage indicated by MAXJ.

H. Setting NPERM > 0.

SILASO allows known eigenpairs to be input directly so that they need not be recomputed. The first column of VAL must contain the eigenvalues (in any order) and the second column of VAL must contain the residual norms ($\|Ay_i - y_i\theta_i\|$, for the eigenpair $\theta_i y_i$). The correct order of magnitude is sufficient. Columns 3 and 4 of VAL are arbitrary. The first NPERM columns of VEC must contain the eigenvectors (which will be orthonormalized by SILASO). The eigenvectors associated with VAL(I,1) must be in the I th column of VEC.

NPERM must be less than or equal to MAXVEC. If NPERM = MAXVEC the program will either confirm that no other desired eigenvalues exist or it will terminate (IERR = -100) as soon as another desired eigenvalue appears.

3. Applicability and Restrictions

SILASO is designed to find all the eigenvalues of a large sparse symmetric matrix lying outside a user defined excluded interval. For small dense matrices, the subroutines provided in EISPACK are to be preferred. It is not necessary for the matrix to be explicitly represented. It is only necessary to provide a subroutine OP to compute matrix-vector products.

In particular SILASO can be combined with a sparse factorization program to compute eigenvalues of A by sectioning. To find all the eigenvalues of A inside an interval [a,b]:

1. Choose a shift $\sigma \varepsilon [a,b]$.
2. Factor $(A - \sigma I) = LDL^T$.
3. Code the subroutine OP to compute $(A - \sigma I)^{-1}x$ by solving $(A - \sigma I)y = x$ using L and D.
4. Set XL = $1/(a - \sigma)$ and XR = $1/(b - \sigma)$.
5. Call SILASO.
6. The eigenvectors returned by SILASO are correct while the eigenvalues must be back transformed as

$$\theta' = 1/\theta + \sigma$$

where θ is the eigenvalue computed by SILASO and θ' is the eigenvalue of A.

Thus it is possible to compute eigenvalues over a wide range by breaking the range into subinterval and sequentially solve for eigenvalues in each subinterval using different shifts σ .

SILASO calls a number of subsidiary functions and subroutines, namely:

SIWLA which implements the block Lanczos algorithm with selective orthogonalization.

SIPPLA which post processes the output of SWLA.

SMVPC which computes residual norms and orthogonality coefficients.

SORTQR which orthonormalizes a block of vectors.

SVZERO which zeroes a given vector.

URAND a FORTRAN IV random number generator given in Forsythe, Malcolm, and Moler [6].

BANDR, BANDV, IMTQL1, IMTQL2, SVD, TRED1, TRED2, and TRIDIB, which are EISPACK Subroutines ([3], [4]).

SAXPY, SCOPY, SDOT, SNRM2, SSCAL and SSWAP which are subset of the BLAS (Basic Linear Algebra Subprograms) written by Lanwson, et. al [7] and modified by Dongarra, et. al [8] for use in LINPACK. If the BLAS are available in assembly language they should be used in place of the FORTRAN IV source code provided.

The user must not use any of the above names in his driver program.

4. Discussion of method and algorithm

The Lanczos algorithm is an efficient scheme for computing a series of vectors q_1, q_2, ... q_j which form an orthonormal basis for the Krylov subspace, span $(q_1, Aq_1, \ldots, A^{j-1}q_1)$.

At each step of the algorithm the Krylov subspace grows larger and one more Lanczos vector is added to the list. The Lanczos algorithm can be interrupted at any step and the Rayleigh-Ritz approximations to eigenpairs of A can be derived from the Krylov subspace quite easily. Thus the algorithm need only continue until the desired approximations are sufficiently accurate.

The block Lanczos algorithm (as described in detail by Underwood [5]) replaces each vector in the simple Lanczos algorithm by an orthornormal block of vectors. Block Lanczos has theoretical advantages over simple Lanczos with respect to finding multiple eigenvalues and has advantages in efficiency if the cost of forming a matrix-vector product is high.

Unfortunately finite precision arithmetic causes the vectors computed by the Lanczos algorithm (both simple and block) to lose orthogonality and approach linear dependence. To maintain robust independence among the Lanczos vectors, SILASO augments the algorithm with selective orthogonalization which causes some of the Lanczos vectors to be orthogonalized against a few selected Ritz vectors, as described in [1] and [2].

The algorithm is terminated when the desired Ritz value are sufficiently accurate. If necessary, SILASO then makes another Lanczos run to test for undisclosed multiplicities. Finally in some cases, SILASO performs a Rayleigh-Ritz procedure on the determined eigenvalues to resolve any clusters.

5. References

[1] B. N. Parlett and D. S. Scott, "The Lanczos Algorithm with Selective Orthogonalization," *Math. of Comp.* 33, 217-38, 1979.

[2] D. S. Scott, "Analysis of the Symmetric Lanczos Process," Ph. D. Thesis, ERL technical report No. M78/40, June 1978, Electronics Research Lab., University of California, Berkeley, CA 94720.

[3] B. T. Smith et al., *Matrix Eigensystem Routines - Eispack Guide*, Lecture Notes in Computer Science 6, 2nd Edition, Springer-Verlag, 1976.

[4] B. S. Garbow, J. M. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines - Eispack Guide Extensions*, Lecture Notes in Computer Science 51, Springer-Verlag, 1977.

[5] R. Underwood, "An Iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems," Ph. D. Thesis, Stanford University, STAN-CS-75-496 (1975).

[6] G. E. Forsythe, M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations*, Series in Automatic Computing, Prentice-Hall.

[7] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for FORTRAN Usage," to appear in *TOMS*.

[8] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *Linpack Users' Guide*, SIAM, 1979.

APPENDIX II

Program Listings

(microfiche, inside back cover)

ORNL/CSD-48
Distribution Category UC-32

INTERNAL DISTRIBUTION

1-2. Central Research Library
3. Patent Office
4. ORNL Technical Library, Document Reference Section
5. Laboratory Records, ORNL R.C.
6-8. Laboratory Records Department
9. A. A. Brooks
10. H. P. Carter/CSD X-10 Library
11. S.-J. Chang
12. L. A. Charlton
13. R. A. Dory
14. R. E. Funderlic
15. P. W. Gaffney
16. D. A. Gardiner
17. M. T. Heath
18-33. D. S. Scott
34. C. A. Serbin
35. A. D. Solomon
36. C. M. Stegall
37. W. C. T. Stoddart
38. R. E. Textor
39. R. C. Ward

EXTERNAL DISTRIBUTION

40. Dr. T. D. Butler, T-3, Hydrodynamics, Los Alamos Scientific Laboratory, P.O. Box 1663, Los Alamos, NM 87545
41. Dr. Bill L. Buzbee, C-1 Applications Support and Research, Los Alamos Scientific Laboratory, P.O. Box 1663, Los Alamos, NM 87545
42. Dr. L. Lynn Cleland, Engineering Research Division, Lawrence Livermore Laboratory, P.O. Box 808, Livermore, CA 94550
43. R. E. Cline, Mathematics Department, University of Tennessee, Knoxville, Tennessee 37916
44. Paul Concus, Computer Science and Applied Mathematics Department, Lawrence Berkeley Laboratory, Berkeley, CA 94720
45. Dr. James Corones, Ames Laboratory, Iowa State University, Ames, IA 50011
46. J. Cullum, IBM T. J. Watson Research Center, Yorktown Heights, NY
47. I. Duff, Computer Science and Systems Division AERE Harwell, England
48. Dr. Marvin D. Erickson, Computer Technology, Systems Department, Pacific Northwest Laboratory, P.O. Box 999, Richland, WA 99352
49. Paul Garabedian, Director, Courant Mathematics and Computing Laboratory, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
50. A. George, Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1
51. David Gilbert, Tennessee Valley Authority, Knoxville, Tennessee 37916
52. Gene H. Golub, Computer Science Department, Stanford University, Stanford, CA 94305

53. Robert T. Gregory, Computer Sciences Department, University of Tennessee, Knoxville TN 37916
54. R. J. Hanson, Numerical Mathematics Division, 5122, Sandia Laboratories, P. O. Box 5800, Albuquerque, NM 87115
55. R. L. Hooper, Statistics, Systems Department, Pacific Northwest Laboratory, P.O. Box 999, Richland, WA 99352
56. Robert Huddleston, Supervisor, Applied Mathematics Division 8325, Sandia Laboratories, Livermore, CA 94550
57. William M. Kahan, Department of Computer Science, University of California, Berkeley, CA 94720
58. R. J. Kee, Applied Mathematics Division, 8331, Sandia Laboratories, Livermore, CA 94550
59. H. B. Keller, Department of Applied Mathematics, 101-50, California Institute of Technology, Pasadena, CA 91109
60. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, New York, NY 10012
61. John Lewis, Department of Mathematical Sciences, John Hopkins University, Baltimore, Maryland 21215
62. R. K. Lohrding, S-1 Statistics, Los Alamos Scientific Laboratory, P.O. Box 1663, Los Alamos, NM 87545
63. P. C. Messina, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL 60439
64. James Ortega, Chairman, Department of Applied Mathematics and Computer Science, University of Virginia, Charlottesville, VA 22903
65. C. C. Paige, School of Computer Science, McGill University, Montreal, Canada
66. B. N. Parlett, Mathematics Department, University of California, Berkeley, CA 94720
67. A. Peshkin, Applied Mathematics Department, Brookhaven National Laboratory, Upton, NY 11973
68. Robert J. Plemmons, Computer Sciences Department, University of Tennessee, Knoxville, TN 37916
69. James C. T. Pool, Office of Basic Energy Sciences, Mail Stop J-309, U.S. Department of Energy, Washington, DC 20545
70. C. Quong, Computer Science and Applied Mathematics Department, Lawrence Berkeley Laboratory, Berkeley, CA 94720
71. Donald Rose, Bell Laboratories, Inc., Murray Hill, NY 07974
72. A. Ruhe, Department of Information Processing, Umea University, S-90187, Umea, Sweden
73. Lawrence F. Shampine, Supervisor, Numerical Mathematics Division, 5122, Sandia Laboratories, P.O. Box 5800, Albuquerque, NM 87115
74. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
75. Richard Underwood, CIS Department, Ohio State University, Columbus, OH 43221
76. C. VanLoan, Computer Science Department, Cornell University, Ithaca, NY 14853
77. Richard S. Varga, Department of Mathematics, Kent State University, Kent, OH 44240
78. B. Wendroff, T-7 Mathematical Modeling & Analysis, Los Alamos Scientific Laboratory, P.O. Box 1663, Los Alamos, NM 87545

79. Olof Widland, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012
80. J. H. Wilkinson, Division of Numerical Analysis and Computer Science, National Physical Laboratory, Teddington, Middlesex TW11 OLW England
81. Office of Energy Research and Development DOE, ORO
82-272. Given distribution as shown in TID-4500 under Mathematics and Computers category (25 copies-NTIS)