Volume 2

Documentation and Listings
Original Lanczos Codes

Lanczos Algorithms for Large Symmetric Eigenvalue Computations

Jane K. Cullum

Ralph A. Willoughby

Originally Published by Birkhäuser-Boston in 1985

- **Volume 1: Theory**

  Volume 1: Theory has been republished verbatim in 2002 as Volume 41 in the SIAM Book Series, Classics in Applied Mathematics. SIAM Publications, Philadelphia, PA. ISBN 0-89871-523-7. (A short Errata for Volume 1 is included in this web version of Volume 2 as Chapter 10.)

- **Uni-Processor and Parallel/Fortran90 Versions**

  Currently, Leonard Hoffnung (Math. Dept, U. Kentucky), Spencer Shellman, (Comp. Sc. Dept, U. Utah) and Jane Cullum (*cullumj@lanl.gov*) are working on uni-processor and on MPI parallel Fortran90 versions of the codes contained in Volume 2. The Hoffnung and Shellman contributions are supported currently by a U.S. Department of Energy, Office of Science, MICS, Los Alamos AMS Program grant. The resulting codes will be made available via the Netlib software repository.

- **Matrix Size**

  This book was published 17 years ago. Computers of today are orders of magnitude faster and have orders of magnitude more memory and storage than those which were availabe when this book was written. Seventeen years ago, a matrix of size $10,000$ was considered very large. Since 1985 some of the algorithms which are included in this book have been used on problems of size a million or more. The requirements are *accurate* matrix computations and *sufficient* computer arithmetic precision.

- **Thanks to Leonard Hoffnung for his help in converting ancient 'script' files for Volume 2.**

Contents

1 Lanczos procedures **1**

1.1 Introduction 1

1.2 What are Lanczos procedures? 4

1.3 Comments and disclaimers 7

2 Real Symmetric Matrices **9**

2.1 Introduction 9

2.2 Documentation for the Codes in Chapters 2, 3, 4, 5 11

2.3 LEVAL: Main Program, Eigenvalue Computations 40

2.4 LEVEC: Main Program, Eigenvector Computations 54

2.5 LEMULT: LANCZS and Sample Matrix-Vector Multiply Subroutines 75

2.6 LESUB: Other Subroutines used by the Codes in Chapters 2, 3, 4, 5 95

2.7 LECOMPAC: Optional Preprocessing Program 124

2.8 LEVAL: LEVEC: File Definitions, Sample Input Files 127

3 Hermitian Matrices **129**

3.1 Introduction 129

3.2 HLEVAL: Main Program, Eigenvalue Computations 132

3.3 HLEVEC: Main Program, Eigenvector Computations 147

3.4 HLEMULT: LANCZS and Sample Matrix-Vector Multiply Subroutines 170

3.5 HLEVAL: HLEVEC: File Definitions, Sample Input Files 190

4 Factored Inverses of Real Symmetric Matrices **193**

4.1 Introduction 193

4.2 LIVAL: Main Program, Eigenvalue Computations 196

4.3 LIVEC: Main Program, Eigenvector Computations . . . 211

4.4 LIMULT: LANCZS and Sample Matrix-Vector Multiply Subroutines . . . 234

4.5 PERMUT: LORDER: LFACT: LTEST: Optional Routines for Chapters 4, 5, 9 . . . 242

4.6 LIVAL: LIVEC: File Definitions, Sample Input Files . . . 263

5 Real Symmetric Generalized Problems 265

5.1 Introduction . . . 265

5.2 LGVAL: Main Program, Eigenvalue Computations . . . 268

5.3 LGVEC: Main Program, Eigenvector Computations . . . 282

5.4 LGMULT: LANCZS and Sample Matrix-Vector Multiply Subnroutines . . . 305

5.5 LGVAL: LGVEC: File Definitions, Sample Input Files . . . 317

6 Real Rectangular Matrices, Singular Values and Vectors 319

6.1 Introduction . . . 319

6.2 Documentation for the Codes in Chapters 6 . . . 322

6.3 LSVAL: Main Program, Eigenvalue Computations . . . 344

6.4 LSVEC: Main Program, Eigenvector Computations . . . 359

6.5 LSMULT: LANCZS and Sample Matrix-Vector Multiply Subroutines . . . 385

6.6 LSSUB: Other Subroutines used by the Codes in Chapter 6 . . . 394

6.7 LSVAL: LSVEC: File Definitions, Sample Input Files . . . 421

7 Nondefective Complex Symmetric Matrices 423

7.1 Introduction . . . 423

7.2 Documentation for the Codes in Chapter 7 . . . 426

7.3 CSLEVAL: Main Program, Eigenvalue Computations . . . 445

7.4 CSLEVEC: Main Program, Eigenvector Computations . . . 458

7.5 CSLEMULT: LANCZS and Sample Matrix-Vector Multiply Subroutines . . . 479

7.6 CSLESUB: Other Subroutines used by the Codes in Chapter 7 . . . 490

7.7 CSLEVAL: CSLEVEC: File Definitions, Sample Input Files . . . 514

8 Real Symmetric Matrices, Block Lanczos Code 517

8.1 Introduction 517

8.2 Documentation for the Codes in Chapters 8 and 9 520

8.3 BLEVAL: Main Program, Eigenvalue and Eigenvector Computations 532

8.4 BLMULT: Sample Matrix-Vector Multiply Subroutines 541

8.5 BLSUB: Other Subroutines used by the Codes in Chapters 8 and 9 547

8.6 BLEVAL: File Definitions, Sample Input File 567

9 Factored Inverses, Real Symmetric Block Lanczos Code 569

9.1 Introduction 569

9.2 BLIEVAL: Main Program, Eigenvalue and Eigenvector Computations 571

9.3 BLIMULT: Sample Matrix-Vector Multiply Subroutines 581

9.4 BLIEVAL: File Definitions, Sample Input File 587

10 Errata: Volume I: Theory 589

Chapter 1

Lanczos procedures

1.1 Introduction

The FORTRAN codes contained in this volume are designed for computing eigenvalues and eigenvectors or singular values and singular vectors of large, sparse matrices. Large means of order several hundred to perhaps 10,000. The largest matrix which we tested was real symmetric and had order 4900. This book is divided into 9 chapters. In this first chapter we give a brief description of Lanczos eigenelement procedures and then make some comments about what the Lanczos codes in this book can and cannot be expected to compute. Detailed analyses of the ideas used in these procedures are contained in Volume 1 of this book.

Chapters 2 through 7 contain procedures which are based upon the single-vector Lanczos recursion with no reorthogonalization of any kind. Six different classes of problems are addressed in these 6 chapters: Eigenelement computations for

1. Real symmetric matrices (Chapter 2)
2. Hermitian matrices (Chapter 3)
3. Factored inverses of real symmetric matrices (Chapter 4)
4. Real symmetric, generalized problems (Chapter 5)
5. Nondefective, complex symmetric matrices (Chapter 7)
6. Singular value and vector computations for real, rectangular matrices (Chapter 6).

Chapters 8 and 9 contain Lanczos procedures which are based upon 'block' versions of the Lanczos recursions. These iterative block procedures include some reorthogonalization within each iteration, but this reorthogonalization is limited to reorthogonalizations w.r.t. certain vectors in each first Lanczos block.

The single-vector procedures can be used to compute anywhere from a very few to very many eigenvalues (singular values). These eigenvalues (singular values) need not be at the extremes of the spectrum. For some matrices it is even possible to compute all of the eigenvalues. The iterative block procedures can only be used to compute a few extreme eigenvalues of the specified matrix. The single vector codes consist of two phases. First eigenvalues or singular values are computed and then corresponding eigenvectors or singular vectors are computed. The iterative 'block' codes compute eigenvalues and corresponding eigenvector approximations simultaneously. Block codes for computing singular values are not included in this book. See for example, Golub, Luk, and Overton [13] for an example of such a block algorithm.

With three exceptions which are given below, each Chapter 2 through 9 contains the following types of information for the particular class of problems considered in that chapter: documentation; main program(s); LANCZS subroutine for computing Lanczos matrices; sample matrix-vector multiply and/or solve subroutines; other subroutines needed by the codes in that chapter; and definitions of the files used by the programs together with sample input files. Because of the similarities between the variables, flags, etc., the documentation for the codes contained in Chapters 2, 3, 4, and 5 was combined and is contained in Section 2.2 of Chapter 2. The codes in Chapters 2, 3, 4, and 5 use essentially (with 2 exceptions) the same set of 'other or additional subroutines' so these subroutines were combined and are given only in Chapter 2, Section 2.6. Similarly, the block codes in Chapters 8 and 9 use the same set of additional subroutines and these are given only in Section 8.5. Some additional optional, preprocessing codes are also provided, and again each of these is included in only one of the chapters and not in each of the ones where it might be useful.

Each set of codes contains many write statements. These write statements serve two major functions: to provide consistency checks on the information supplied by the user, and to provide running commentary on the progress of the computations. Much of the code has been modularized to help make the program logic more transparent to the user. These codes are not designed as efficiently as they could be. Many internal comments have been included. Numerous consistency checks have been used to verify that the user has set up the procedure properly. Basically, we have compromised some efficiency for safety and robustness.

Each LANCZS subroutine together with the corresponding sample matrix-vector multiply and solve subroutines are in files labelled as *MULT. For example in Chapter 2 where real symmetric matrices are discussed this file is labelled LEMULT. The user should note that within a given *MULT file, each sample USPEC* and *MATV subroutine has been given two names so that these subroutines can co-exist with similar subroutines for other test matrices. However, two different *MULT files cannot co-exist because subroutine names are reused in going from one category of matrices to another category. In particular for the codes in Chapters 2, 3, and 7, the matrix-vector multiply subroutine is called CMATV. Moreover, in all of the chapters, the matrix specification subroutines are called USPEC. This reuse of names makes it easier for the user to pass from one set of codes to another. Furthermore, from category to category, subroutines with similar function were typically given the same name. For example, all of the subroutines which generate families of Lanczos matrices are named LANCZS. There are two BISEC bisection subroutines for computing eigenvalues of real symmetric tridiagonal matrices, one for Chapters 2, 3, 4, and 5 and the other one is for Chapter 6. If these sets of codes had to co-exist in one computer file, then it would be necessary for the user to devise a scheme for renaming those subroutines which have the same names.

With respect to portability, each of these programs and subroutines has been individually checked for portability by the PFORT Verifier [22], but the communications between these subroutines have not been checked. Obvious problems with portability like non-Fortran items in the format statements have all been removed. However, certain nonportable constructions have been retained because they make the programs somewhat easier to use. The header of each of the programs contains a list of those constructions in that program which were identified by the PFORT verifier as being nonportable. These headers can be used to locate the nonportable items so that if necessary they can be modified. A list of most of the nonportable items and the reasons for retaining them are given in Table 1.1.

The single vector Lanczos codes in Chapters 2 through 7 are essentially self-contained. The user must provide the matrix-vector multiply and/or solve subroutines which are required by these codes, together with a matrix specification subroutine which defines, dimensions and initializes the matrix which will be used by the Lanczos procedure. The sample matrix-specification subroutines and sample matrix-vector multiply and solve subroutines contained with these codes can be modified and used if appropriate or they can be replaced completely. All of these procedures require a random number generator subroutine, inner product subroutines, and a subroutine to mask underflow. These procedures assume that each time the random number generator is called that the seed for this generator is automatically reset to a different value.

The iterative 'block' Lanczos codes in Chapters 8 and 9 require matrix specification and matrix-vector

Nonportable Construction	Where Used	Why Used
Entry	Passes storage locations of arrays and parameters needed to define user-specified matrix from subroutine USPEC where arrays are dimensioned and initialized to the corresponding matrix-vector multiply or solve subroutine.	Codes do not need to 'see' the user-specified matrix. Codes need only output from matrix-vector multiply or solve subroutines for the matrix being used. User does not have to alter the calling sequences to these subroutines every time the number or kind of arrays needed to define the given matrix is changed.
Formats (20A4) and (4Z20)	(20A4) is used to read and write explanatory comments within the main programs and in sample USPEC subroutines.	Allows the user to easily modify headers describing the matrix and code being used.
	Machine format (4Z20) is used to read in and write out the Lanczos tridiagonal matrices generated and other quantities for which conversion errors could cause numerical problems.	Prevents format conversion errors incurred in input/output conversions.
Free Format Read (5,*)	Used in main program and in sample USPEC subroutines on read-ins of user-specified parameters from input file 5.	Ease of input. User does not have to have the input values properly aligned in the input file.
Complex*16 Variables	Used only in the Hermitian and in the complex symmetric Lanczos codes.	Computations require double precision complex arithmetic.
Specification of Machine Epsilon	Used in main programs	Required to define tolerances used at various points in the computations.

Table 1.1: Nonportable Constructions Used in the Codes

multiply and solve subroutines very similar to those used in the single vector codes, plus the same type of random number generating subroutine, inner product subroutine, and mask subroutine. However, as implemented here the block codes are not self-contained. These codes call two subroutines from the EISPACK Library [23, 8], TRED2 and IMTQL2, which are used repeatedly to compute the eigenvalues and eigenvectors of the small Lanczos matrices generated on each iteration of the block procedures. The user can of course replace these calls by calls to subroutines which perform similar functions, if the EISPACK Library is not available.

The optional preprocessing programs in Sections 2.7, 4.5, 6.7, and 7.7 are stand-alone (if one includes the programs which must be supplied by the user), except for the subroutine PERMUT given in Section 4.5. PERMUT can be used in conjunction with the procedures in Chapters 4, 5, and 9. It calls the SPARSPAK Library [9] (A. George, J. Liu, E. Ng, U. Waterloo) to try to determine a reordering of the given sparse matrix for which the sparsity of the given matrix translates into a sparse factorization of the reordered matrix.

1.2 What are Lanczos procedures?

Lanczos procedures for computing eigenvalues and eigenvectors of real symmetric matrices are based upon one or more variants of the basic single-vector Lanczos recursion for tridiagonalizing a real symmetric matrix A. Given a starting vector v_1 which is typically-generated randomly, the Lanczos recursion implements a Gram-Schmidt orthogonalization of the matrix-vector products Av_i corresponding to the Lanczos vectors v_i generated by the recursion. See for example Bjorck [1]. Specifically, we have that for $i = 2, \ldots, m$,

$$\beta_{i+1} v_{i+1} = A v_i - \alpha_i v_i - \beta_i v_{i-1} \tag{1.2.1}$$

where $\alpha_i \equiv v_i^T A v_i$ and $\beta_{i+1} \equiv v_{i+1}^T A v_i$. By definition $\alpha_i v_i$ and $\beta_i v_{i-1}$ are the projections of Av_i onto the two most recently-generated Lanczos vectors v_i and v_{i-1}. In practice to improve the numerical stability of this recursion, the above formulas are replaced by the following ones.

$$\alpha_i \equiv v_i^T (A v_i - \beta_i v_{i-1}) \text{ and } \beta_{i+1} \equiv \|A v_i - \alpha_i v_i - \beta_i v_{i-1}\|. \tag{1.2.2}$$

The α_i as defined in Eqn(1.2.2) correspond to a modified Gram-Schmidt orthogonalization procedure. The formula for β_{i+1} given in Eqn(1.2.2) is theoretically equivalent to the one given with Eqn(1.2.1). However, it is superior numerically because this choice directly controls the sizes of the Lanczos vectors. See Paige [19].

Rewriting Eqn(1.2.1) in matrix form, we obtain

$$A V_j = V_j T_j + \beta_{j+1} v_{j+1} e_j^T \tag{1.2.3}$$

where T_j denotes the real symmetric tridiagonal Lanczos matrix of order j whose diagonal entries are the scalars α_i, $1 \le i \le j$, and whose subdiagonal (superdiagonal) entries are the scalars β_{i+1}, $1 \le i \le j-1$, generated by the Lanczos recursion. In Eqn(1.2.3), $V_j = (v_1, v_2, \ldots, v_j)$, the matrix whose columns are the Lanczos vectors generated by the recursion, and e_j is the coordinate vector whose j-th component is 1 and whose other components are 0.

It is easy to demonstrate by induction that in exact arithmetic each set of vectors V_j generated by the recursion in Eqns(1.2.1) and (1.2.2) is an orthonormal set. Therefore for any A-matrix with n distinct eigenvalues and any starting vector v_1 which has a projection on every eigenspace of A, we have that for each $j \le n$,

$$T_j = V_j^T A V_j. \tag{1.2.4}$$

Thus the symmetric tridiagonal matrices T_j are representations of the projections of the given matrix A onto the subspaces spanned by the corresponding sets of Lanczos vectors V_j. The eigenvalues of these matrices are the eigenvalues of the A-matrix restricted to these subspaces. Since the Lanczos vectors are obtained by orthogonalizing vectors of the form $\{v_1, Av_1, A^2v_1, \ldots,\}$, we expect the eigenvalues of the T_j to provide good approximations to some of the eigenvalues of A, if j is sufficiently large. Clearly, at least theoretically, if we extend the recursion to $j = n$, then the eigenvalues of T_n will be the eigenvalues of A. T_n is simply an orthogonal transformation of A and must therefore have the same eigenvalues as A. Moreover, any Ritz vector $V_j u$ obtained from an eigenvector u of some T_j is an approximation to a corresponding eigenvector of A.

Basic steps in any Lanczos procedure for computing eigenvalues and eigenvectors of 'symmetric' matrices are the following.

1. Use a variant of the Lanczos recursion to transform the given 'symmetric' matrix A into a family of 'symmetric' tridiagonal matrices of varying sizes.
2. Compute eigenvalues and eigenvectors of certain members of this family. Because of the real symmetric tridiagonal structure this is a much simpler problem than computing the eigenvalues and eigenvectors of A directly.
3. Take some or all of these eigenvalues as approximations to eigenvalues of A and map the corresponding eigenvectors of the tridiagonal matrix into Ritz vectors for the matrix A.
4. Use these Ritz vectors as approximations to the eigenvectors of A.

The Lanczos recursion in Eqn(1.2.1) has several properties which make it particularly attractive for dealing with large but sparse matrices. First the given matrix enters the recursion only through the matrix-vector multiply terms Av_i. Thus contrary to what is done in the standard methods for solving small or medium size eigenvalue problems, see for example EISPACK [23, 8], the given matrix is not explicitly modified. The user must provide only a subroutine which computes Ax for any given vector x. If the matrix A is sparse, this computation can be done using an amount of storage that is only linear in the size of the matrix instead of quadratic. Second, the recursion uses only the two most recently-generated Lanczos vectors. The Gram-Schmidt orthogonalization of an arbitrary set of vectors would require that at any given stage in the process that all of the vectors which have already been orthogonalized be available for orthogonalizing each additional vector as it is considered. Thus, the storage requirements for implementing the basic Lanczos recursion are minimal. If we use Eqns(1.2.1) and (1.2.2) then only 2 n-vectors are needed for the two most recently-generated Lanczos vectors plus storage for the α and β arrays.

There are however numerical problems if only a simple direct implementation of this recursion is programmed. In general such an implementation yields Lanczos matrices which have extra eigenvalues in addition to the 'good' eigenvalues which are approximations to eigenvalues of A. These extraneous or 'spurious' eigenvalues are caused by the losses in the orthogonality of the Lanczos vectors which in turn are caused by the combination of the roundoff errors resulting from the finite computer arithmetic and the convergence (as j is increased) of eigenvalues of the Lanczos matrices to eigenvalues of the original matrix A. This interaction between the computer arithmetic and the convergence of eigenvalues is discussed in Paige [17, 20].

During the past $5 - 10$ years many different types of Lanczos eigenelement algorithms have been proposed. See Volume 1, Chapter 2 of this book for a brief survey of the literature. Most of these procedures incorporate modifications to the basic Lanczos recursion in Eqns(1.2.1) and (1.2.2) which force the Lanczos vectors to stay nearly orthonormal. These approaches require either the repeated computation of Ritz vectors or the repeated reorthogonalization of the Lanczos vectors as they are generated or some combination of these two computations. In either case as the size of the Lanczos matrix generated is increased to be able to compute more eigenvalues, the associated Ritz vectors or the Lanczos vectors needed for the reorthogonalizations require more and more storage. These modifications often work well but destroy much of the simplicity of the basic procedure, and because of the added storage requirements resulting from the reorthogonalizations they limit the number of eigenelements which can be computed.

The approach which we have chosen and which is implemented in the enclosed FORTRAN programs in Chapters 2 through 7 is not to force the orthogonality of the Lanczos vectors by reorthogonalizing, but to work directly with the basic Lanczos recursion, accepting the losses in orthogonality, and then unraveling the effects of these losses. This approach allows us to retain the basic simplicity of the Lanczos recursion, to minimize the storage requirements, and to therefore maximize the number of eigenvalues of A which can be computed. In our approach in the single-vector algorithms in Chapters 2 through 7, Ritz vectors are not computed until after the eigenvalues have been computed accurately. Consequently, the basic storage requirements for our eigenvalue (singular value) algorithms are only a small multiple of the size of the largest Lanczos matrix used in the computations. Thus, we can compute many eigenvalues of very large but sparse matrices. Depending upon what is to be computed and upon the eigenvalue distribution in the given matrix A, the sizes of the Lanczos matrices used in these computations may be much smaller or considerably larger than the original A-matrix. However the Lanczos matrices generated by the procedures in Chapters 2 through 6 are real symmetric and tridiagonal so that these matrices can be very large and still not present insurmountable computational problems. Eigenvalue and eigenvector computations for such matrices require minimal amounts of storage and fairly reasonable numbers of arithmetic operations.

The computational problems which arise from not maintaining near orthogonality of the Lanczos vectors and which we must address in our single-vector codes are of two types. First and most importantly, we must deal with the question of sorting the eigenvalues of the Lanczos matrices into 2 classes, one corresponding to the 'good' eigenvalues which are approximations to the eigenvalues of A and the other corresponding to the extra or 'spurious' eigenvalues caused by the losses in orthogonality. The identification test used for doing this is discussed in Volume 1, Chapter 4, Section 4.5. For the procedures discussed in Chapters 2 through 6, this identification test is an integral and inexpensive part of the eigenvalue (singular value) computations. For the complex symmetric procedure discussed in Chapter 7 this test is handled in a considerably less eloquent manner and is expensive.

The second but much less serious difficulty we must address is the question of false multiplicities. The multiplicity of a particular 'good' eigenvalue as an eigenvalue of the Lanczos matrices is not related to the multiplicity of that eigenvalue as an eigenvalue of the A-matrix. 'Good' eigenvalues may replicate many times as eigenvalues of a Lanczos matrix, but be only simple eigenvalues of the original A-matrix. Thus, these single-vector procedures cannot directly determine the true multiplicities of the computed 'good' eigenvalues. Of course, this latter comment is also applicable to any single-vector Lanczos procedure not just to our procedures. Theoretically, at most one eigenvector for each distinct eigenvalue of the A-matrix can be obtained using the single-vector Lanczos recursion given in Eqns(1.2.1) and (1.2.2). (This of course is not true for iterative block Lanczos procedures.) It is interesting to note however that if the Lanczos recursion is used without any reorthogonalization, then it can yield sets of linearly independent eigenvectors for eigenvalues which are multiple in the A-matrix. The amount of work required to compute these additional eigenvectors depends upon the particular matrix in question and upon the particular eigenvalue. The codes provided in Chapters 2 through 7 of this book do not however incorporate this capability.

The iterative 'block' Lanczos procedures for real symmetric matrices given in Chapters 8 through 9 are based upon a block version of the Lanczos recursion

$$Q_{j+1}B_{j+1} = AQ_j - Q_jA_j - Q_{j-1}B_j^T \tag{1.2.5}$$

for $j = 1, 2, \ldots, s$ where Q_1 is $n \times q$ and the coefficient matrices A_j and B_{j+1} are block analogs of the scalar coefficients in the single-vector Lanczos recursion in Eqns(1.2.1) and (1.2.2). The number of blocks s used on each iteration is chosen such that $qs \ll n$, where n is the order of the given A-matrix and q is chosen such that $q \geq q'$, the number of eigenvalues and eigenvectors desired. The Lanczos matrices are real symmetric, block tridiagonal matrices. In Eqn(1.2.5) we used Q_j instead of V_j because in our block Lanczos procedures we maintain near-orthogonality of the blocks generated within each iteration by incorporating reorthogonalization of the blocks of Lanczos vectors with respect to certain vectors in the first Lanczos block.

The 'block' procedures provided in Chapters 8 and 9 are really hybrid algorithms, something between a true block Lanczos procedure, see for example, Cullum and Donath [4, 3] and Chapter 7 in [5], and the single-vector Lanczos procedures given in Chapters 2 through 7. The sequence of 'blocks' generated on each iteration of this hybrid method has the property that the first Q-block contains at least as many vectors as the user is trying to compute, but the second and succeeding blocks each contain only one vector. The corresponding resulting Lanczos matrices are not block tridiagonal. Each Lanczos matrix has a border of blocks in the first q rows and columns and is tridiagonal below this border.

At the beginning of each chapter, a brief description is given of the particular variant of the Lanczos recursion used in the Lanczos codes included in that chapter, along with some additional comments relevant to the particular types of problems being considered in that chapter.

1.3 Comments and disclaimers

The single-vector Lanczos procedures contained in Chapters 2 through 7 do not behave like standard eigenelement procedures. Their behavior is both non-classical and somewhat unorthodox. If one of these codes were run on two different kinds of computers but with the same original matrix and the same initial specifications, the computed results could be quite different. A primary cause for such differences can of course be a difference in the starting vector caused by a difference in the random number generators. However even if the same starting vector were read in, the results would almost surely differ due to the differences in the computer arithmetic. In practice, the Lanczos matrices generated on two different kinds of computers may agree for a certain number of Lanczos steps but will begin to diverge upon the convergence of one or more of the eigenvalues of these Lanczos matrices to eigenvalues of the A-matrix. If after a reasonable number of steps in the Lanczos recursion we were to compare the entries in the Lanczos matrices generated by the two different computers, the values would probably be very different.

Furthermore, if we were to compute the eigenvalues of the two sets of Lanczos matrices for various sizes and 'spurious' eigenvalues were present, then these spurious eigenvalues would be different and even appear in different portions of the spectrum. In fact, prior to the convergence of a particular 'good' eigenvalue, the values of that good eigenvalue, in terms of how accurate it is at any given stage in the computations, may differ. However once a 'good' eigenvalue in either set has converged, that 'good' eigenvalue will agree with a true eigenvalue of the original user-specified matrix to as many digits as can be expected.

Therefore, if the user carries out the sample eigenvalue computation provided in Chapter 2, he/she should not be alarmed or surprised if the output from the computer being used does not agree with what is shown in the sample, as long as the converged 'good' eigenvalues agree. Actually one may observe different rates of convergence on different kinds of computers, depending upon the computer arithmetic. With increased arithmetic precision in all of the computations, these procedures may converge more rapidly. With decreased precision, they will converge less rapidly. All of our codes use double precision arithmetic (for an IBM 3083) and any precision less than that is not recommended.

Each of these procedures requires the user to supply either a matrix-vector multiply subroutine or a matrix-vector solve subroutine. (Both types of subroutines are required for the codes in Chapter 5 .) Such subroutines should perform the required computations rapidly and accurately, taking advantage of any special properties or structure in the given matrix. Our Lanczos programs see the original matrix as the outputs of these subroutines. The codes provided include sample matrix-vector multiply subroutines for a general sparse 'symmetric' matrix given in a particular sparse format. These are available for the user to use or modify as desired. Note that similar programs are also provided for the singular value/vector computations. Accuracy is important in these subroutines because consistency must be maintained in the information being provided to the LANCZS subroutine which is generating the Lanczos matrices. There is no built-in mechanism for preserving symmetry. Therefore, the matrix-vector multiply and solve subroutines must be coded with care. Without such consistency the Lanczos codes will not function properly.

The convergence characteristics of the two types of Lanczos procedures considered are quite different. These differences are discussed in Chapters 4 and 7 of Volume 1 of this book. However, in both cases, the degree of difficulty in computing the desired eigenvalues depends upon the eigenvalue gaps. For the single-vector procedures the primary factor in determining whether or not it is feasible to compute either large numbers of eigenvalues or the eigenvalues with the smallest gaps, is the gap ratio, the ratio of the largest gap between two neighboring eigenvalues to the smallest such gap. The smaller this ratio, the easier it is to compute all of the eigenvalues of the given matrix. The larger this ratio, the harder it is to compute those eigenvalues with the smallest gaps. The locations of the desired eigenvalues in the spectrum of the given matrix also play a significant role in the rate of convergence of individual eigenvalues. Both types of Lanczos procedures favor extreme eigenvalues. The iterative block codes, in fact, can only compute a few extreme eigenvalues. However for the single-vector codes, it is possible for interior eigenvalues which have gaps which are significantly larger than the gaps for some of the extreme eigenvalues to converge prior to the convergence of those extreme eigenvalues. Examples of the convergence achievable are given in Volume 1, Chapter 4 of this book.

The convergence of the iterative block procedures depends primarily upon the gaps between the eigenvalues being computed and the closest eigenvalue not being approximated, the spread of the matrix, and the overall eigenvalue distribution. The block procedures discussed in Chapters 8 and 9 are iterative and the codes track the rate of convergence. If the observed rate is too slow (as specified by the user), these block procedures will terminate without achieving convergence. The user then has the option of restarting the block procedure with a different choice of parameters and using the current approximation to the basis for the desired eigenspace as the starting vectors.

Thus, the amount of work required for a particular eigenelement computation for a given matrix using a particular method depends directly upon the eigenvalue distribution in that matrix and upon which portion of the spectrum is being computed. Some problems are ‘easy’, others are hard. Therefore failure can occur, in the sense that these procedures may not be able to compute the information desired by the user within the computational bounds specified by the user. However the single-vector Lanczos procedures, even in ‘failure’, provide a great deal of information about the eigenvalue spectrum of the given matrix.

In deciding which procedure to use on a given problem, our preference is a single-vector procedure, although the iterative block procedures can often quickly provide simultaneously the desired eigenvalues and eigenvectors. If the user wants extreme eigenvalues and the user knows or suspects that one or more of these is multiple, then the block procedure is probably preferable. More details about the Lanczos procedures contained in this book can be found in Volume 1. Any questions about these programs including the question of obtaining copies of these codes or of problems with these codes, should be addressed directly to the authors. We hope that these codes will prove useful in many different applications in the engineering and scientific community.