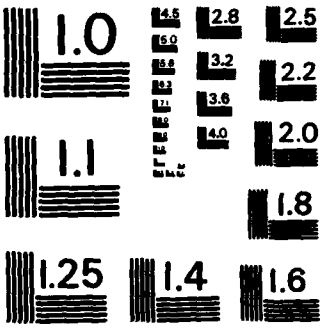
AD-A160 210 LANCZOS ALGORITHM APPLIED TO MODAL ANALYSIS OF VERY LARGE STRUCTURES(U) CALIFORNIA UNIV BERKELEY DEPT OF MATHEMATICS B N PARLETT ET AL. 10 AUG 85 1/1

UNCLASSIFIED AFOSR-TR-85-0795 F49620-84-C-0090 F/G 12/1 NL

END

FILMED

DTIC



SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR-TR- 85-0795
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Lanczos Algorithm Applied to Modal Analysis of Very Large Structures

5. TYPE OF REPORT & PERIOD COVERED

Final 8-1-84 to 8-1-85

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

B.N. Parlett, P.S. Jensen, T. Erickson

8. CONTRACT OR GRANT NUMBER(s)

F49620-84-C-0090

9. PERFORMING ORGANIZATION NAME AND ADDRESS

University of California
Dept of Math
Berkely CA 94720

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

61102F 2304 A8

11. CONTROLLING OFFICE NAME AND ADDRESS

Air Force Office of Scientific Research
Bldg. 410 Bolling AFB, DC 20332

12. REPORT DATE

1 Aug 1985

13. NUMBER OF PAGES

x

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)

Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

DTIC ELECTE OCT 11 1985 D A

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Eigenanalysis; Symmetric Matrices; Large Sparse Matrices; Structural analysis; Lanczos; Modal ~~Analisys~~ Analysis; Generalized eigenproblems.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

x

AD-A160 210

DTIC FILE COPY

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

INSTRUCTIONS FOR PREPARATION OF REPORT DOCUMENTATION PAGE

RESPONSIBILITY. The controlling DoD office will be responsible for completion of the Report Documentation Page, DD Form 1473, in all technical reports prepared by or for DoD organizations.

CLASSIFICATION. Since this Report Documentation Page, DD Form 1473, is used in preparing announcements, bibliographies, and data banks, it should be unclassified if possible. If a classification is required, identify the classified items on the page by the appropriate symbol.

COMPLETION GUIDE

General. Make Blocks 1, 4, 5, 6, 7, 11, 13, 15, and 16 agree with the corresponding information on the report cover. Leave Blocks 2 and 3 blank.

Block 1. Report Number. Enter the unique alphanumeric report number shown on the cover.

Block 2. Government Accession No. Leave Blank. This space is for use by the Defense Documentation Center.

Block 3. Recipient's Catalog Number. Leave blank. This space is for the use of the report recipient to assist in future retrieval of the document.

Block 4. Title and Subtitle. Enter the title in all capital letters exactly as it appears on the publication. Titles should be unclassified whenever possible. Write out the English equivalent for Greek letters and mathematical symbols in the title (see *"Abstracting Scientific and Technical Reports of Defense-sponsored RDT/E,"AD-667 000*). If the report has a subtitle, this subtitle should follow the main title, be separated by a comma or semicolon if appropriate, and be initially capitalized. If a publication has a title in a foreign language, translate the title into English and follow the English translation with the title in the original language. Make every effort to simplify the title before publication.

Block 5. Type of Report and Period Covered. Indicate here whether report is interim, final, etc., and, if applicable, inclusive dates of period covered, such as the life of a contract covered in a final contractor report.

Block 6. Performing Organization Report Number. Only numbers other than the official report number shown in Block 1, such as series numbers for in-house reports or a contractor/grantee number assigned by him, will be placed in this space. If no such numbers are used, leave this space blank.

Block 7. Author(s). Include corresponding information from the report cover. Give the name(s) of the author(s) in conventional order (*for example, John R. Doe or, if author prefers, J. Robert Doe*). In addition, list the affiliation of an author if it differs from that of the performing organization.

Block 8. Contract or Grant Number(s). For a contractor or grantee report, enter the complete contract or grant number(s) under which the work reported was accomplished. Leave blank in in-house reports.

Block 9. Performing Organization Name and Address. For in-house reports enter the name and address, including office symbol, of the performing activity. For contractor or grantee reports enter the name and address of the contractor or grantee who prepared the report and identify the appropriate corporate division, school, laboratory, etc., of the author. List city, state, and ZIP Code.

Block 10. Program Element, Project, Task Area, and Work Unit Numbers. Enter here the number code from the applicable Department of Defense form, such as the DD Form 1498, "Research and Technology Work Unit Summary" or the DD Form 1634, "Research and Development Planning Summary," which identifies the program element, project, task area, and work unit or equivalent under which the work was authorized.

Block 11. Controlling Office Name and Address. Enter the full, official name and address, including office symbol, of the controlling office. (*Equates to funding/sponsoring agency. For definition see DoD Directive 5200.20, "Distribution Statements on Technical Documents."*)

Block 12. Report Date. Enter here the day, month, and year or month and year as shown on the cover.

Block 13. Number of Pages. Enter the total number of pages.

Block 14. Monitoring Agency Name and Address (*if different from Controlling Office*). For use when the controlling or funding office does not directly administer a project, contract, or grant, but delegates the administrative responsibility to another organization.

Blocks 15 & 15a. Security Classification of the Report: Declassification/Downgrading Schedule of the Report. Enter in 15 the highest classification of the report. If appropriate, enter in 15a the declassification/downgrading schedule of the report, using the abbreviations for declassification/downgrading schedules listed in paragraph 4-207 of DoD 5200.1-R.

Block 16. Distribution Statement of the Report. Insert here the applicable distribution statement of the report from DoD Directive 5200.20, "Distribution Statements on Technical Documents."

Block 17. Distribution Statement (*of the abstract entered in Block 20, if different from the distribution statement of the report*). Insert here the applicable distribution statement of the abstract from DoD Directive 5200.20, "Distribution Statements on Technical Documents."

Block 18. Supplementary Notes. Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of (*or by*) . . . Presented at conference of . . . To be published in . . .

Block 19. Key Words. Select terms or short phrases that identify the principal subjects covered in the report, and are sufficiently specific and precise to be used as index entries for cataloging, conforming to standard terminology. The DoD "Thesaurus of Engineering and Scientific Terms" (*TEST*), AD-672 000, can be helpful.

Block 20. Abstract. The abstract should be a brief (*not to exceed 200 words*) factual summary of the most significant information contained in the report. If possible, the abstract of a classified report should be unclassified and the abstract to an unclassified report should consist of publicly- releasable information. If the report contains a significant bibliography or literature survey, mention it here. For information on preparing abstracts see "Abstracting Scientific and Technical Reports of Defense-Sponsored RDT&E," AD-667 000.

★ U.S. GOVERNMENT PRINTING OFFICE : 1973-729-091 '1431 3-1

Report II

~~Detailed Study of the~~ Lanczos Algorithm Applied to Modal Analysis of very large Structures

Final Report

10 August 1985

Contract F49620-84-C-0090

B.N. Parlett
Center for Pure and Applied Mathematics
Universtiy of California
Berkeley, CA 94720
(415) 642-6655

Foreword

This report was prepared by the University of California in fulfillment of the requirements of Contract F49620-84-C-0090 with the Air Force Office of Scientific Research. The project manager for this contract was Capt. John Thomas of AFOSR.

The technical work was performed during the period August 1984 through July 1985. The technical efford involved development of simple and block Lanczos algorithms at the University of California and testing of these algorithms for modal analysis of typical structural analysis problems at Lockheed Missiles and Space Company, Inc. Work at the University of California was carried out by Drs. T. Ericsson, B. Nour-Omid and Professor B.N. Parlett under the direction of Prof. Parlett. Work at Lockheed Missiles and Space Company was carried out by Paul S. Jensen under contract to the University of California.

One report is being prepared as a result of this research for publication in Math. Comp. It is entitled "How to Implement the Spectral Transformation," by all four participants in this project.

The contract was approved too late to provide support for Dr. Nour-Omid who left the Center for Pure and Applied Math. in April 1984. The surplus funds went in partial support of Dr. Ericsson.

INTRODUCTION

B.N. Parlett
Aug. 6, 1985

Research over the past 9 years at the University of California has resulted in a number of reports.
[1-10] on the general theory of the Lanczos algorithm for large symmetric eigenproblems. This work was extended in [5] to apply to the large generalized symmetric eigenanalysis problem. The purpose of this effort was to implement the results of the past research for application to large generalized problems arising in structural analysis.

1.0 Simple Lanczos

The first implementation developed was a simple Lanczos algorithm based on selective orthogonalization [3 and 9] and a new technique [11] for monitoring eigenvalues of the intermediate tridiagonal matrices T_j.

These are our approximate eigenvalues. At the same time we get error bounds. By monitoring at every step the Lanczos run may be terminated as early as possible. This poses the challenge of carrying out this monitoring so effectively that it increases the cost of a Lanczos step by less than 5%, preferably by less than 1%. This, in turn, means updating a few eigenvalues of a $j \times j$ tridiagonal in $O(j)$ arithmetic operations. For comparisons we note that the EISPACK program IMTQL1 requires about $9j^2$ to compute all eigenvalues. Our solution is given in [11]

After considerable testing on the local VAX/UNIX system this simple Lanczos implementation LANSO was sent to Lockheed and was up and running by Sept. 1984 (See Sec.2). By October troubles became apparent. For short runs of 20/25 steps, picking up 7 or 8 eigenvalues, all seemed well. However the code regularly malfunctioned a little later, certainly before Step 50.

A serious tactical error in the code was the decision to monitor not more than 8 unconverged eigenvalues per step. This value seemed effective in the tests at Berkeley. It was only in June 1985 that we realized that this ceiling was provoking most of the difficulties with LANSO. The easy change from 8 to 16 permitted runs of up to 100 steps. It so happened that 6 slowly converging eigenvalues blocked from view eigenvalues that were converging rapidly.

Here is the great value of this collaborative research effort. Without Lockheed's difficult problems this design defect would not have been uncovered.

1.1 Blcok Lanczos

The block Lanczos implementation BLANSO was sent to Lockheed in October 1984 and was running by January, 1985.

All but one aspect of the simple Lanczos program generalize readily, but expensively, to the block case. The intermediate matrix T is now block tridiagonal and it is proving difficult to generalize the program ANALYZET for monitoring the convergence of T's eigenvalues. As an interim measure we are using EISPACK codes but these are too general and too expensive for long runs. Not only are eigenvalues needed but the last section of the associated eigenvectors.

Our research on this topic is described by Dr. T. Ericsson in Section 3.

1.2 Singular M.

Our investigations revealed an interesting problem to which no attention has been given in the literature. What happens when the inertia Matrix M is positive semi-definite but deficient in rank? This feature of M occurs quite frequently, especially in the analysis of structures intended for service in space. We had assumed that no difficulties would arise. Indeed if all vectors are projected onto the subspace associated with the finite eigenvalues then no difficulties do arise. What we realized, albeit gradually, is that every vector is contaminated by components in the null space of M and these components grow steadily. One reason that this phenomenon may have lain dormant for so long is that it is difficult to detect. All our usual measures involve M and look fine because M blanks out these unwanted components. A consequence of this feature is that computed results turn out to be less accurate than expected. The phenomenon is not special to our codes. It will affect all implementations based on direct iteration. There is a simple way to cure this misbehavior but it is very expensive. One simply orthogonalizes the computed eigenvectors against the null space of M but using the K-inner product.

However we have found a very inexpensive procedure and it seems to be satisfactory. At each step there is a residual vector that is not used explicitly in the Lanczos process. This vector will become the next Lanczos vector. It turns out that a modification used by Ruhe and Ericsson for another purpose does, indirectly, suppress the unwanted components in the computed Ritz vecotrs. The block version is mentioned in §§1.2 of Section 2 (by P.S. Jensen).

All this is a somewhat unexpected dividend of our investigations.

1.3 Termination Criteria.

As we began to make longer Lanczos runs we found that the convergence rate, on our cases, was remarkably good. In one run of 100 Lanczos steps 68 eigenvalues converged to working (double) precision. We had previously thought that a 2:1 ratio was impressive.

This perception brings to the fore a difficult decision that users must make. For how many steps should a Lanczos iteration be continued before a new shift is chosen and a new factorization of K-(shift) M is made? Ericsson and Ruhe [5] favored many short runs (of length 20) but our investigations suggest the opposite. It is cost effective to keep going whilever Ritz values are stabilizing at a good rate. More work is needed here.

REFERENCES

1. C.C. Paige, "Computational Variants of the Lanczos method for the eigenproblem," J. Inst. Math. Appl., vol.10, (1972) 373-381.

2. "An analysis of Lanczos algorithms for symmetric matrices" (with W. Kahan). Memorandum No. ERL-M467, UC Berkeley, September 1974, 49 pp.

3. "The Lanczos algorithm with selective orthogonalization" (with D.S. Scott). Mathematics of Computation, vol. 33 (1979), pp.217-238.

4. "The symmetric eigenvalue problem" (Prentice Hall, New Jersey, 1980 348 pp.)

5. T. Ericsson and A. Ruhe, "The Spectral Transformation Lanczos Method in the Numerical Solution of Large, Sparse, Generalized, Symmetric Eigenvalue Problems," Math. Comp. 34 (1980) 1251-1268.

6. "Tracking the progress of the Lanczos Algorithm for large symmetric Eigen Problems," (with J.K. Reid, IMA Journ. of Numerical Analysis, vol. 1 pp. 135-155, (1981).

7. "On estimating the largest eigenvalue with the Lanczos algorithm" (with H. Simon and L.M. Stringer), Math. Comp. 38 pp 153-165 (1982).

8. "Lanczos versus subspace iteration for solution of Eigenvalue Problems," (with Nour-Omid and Taylor) Internat. J. for Num. Meths. in Eng. vol 19 (1983) pp.859-871.

9. H. Simon, "The Lanczos Algorithm with Partial Reorthogonalization," Math. Comp., vol.42 (1984) 115-142.

10. J. Cullum and R.A. Willoughby, "Lanczos Methods for Large Symmetric Eigenvalue Problems, Birkhauser-Boston (1984).

11. B.N. Parlett and B. Nour-Omid, "The Use of Refined Error Bounds when updating eigenvalues of tridiagonals," J. Lin. Alg. Appl. vol 68, (1980) 179-220.

Items 2,3,4,6,7,8 were written by B.N. Parlett either alone or with the indicated coworkers.

TEST RESULTS FOR THE LANCZOS ALGORITHM STUDY

FINAL REPORT

Paul S. Jensen
29 July 1985

Work done for: University of California
2405 Bowditch St.
Berkeley, CA 94720

Cust.order: 5-432997-MA

Lockheed Palo Alto Research Laboratory
3251 Hanover Street, Orgn. 9250, Bldg.255
Palo Alto, California 94304

Section 2. TEST RESULTS FOR THE LANCZOS ALGORITHM STUDY

Paul S. Jensen 29 July 1985

This study concerns two computer implementations of the Lanczos algorithm developed under AFOSR contract F49620-84-C-0090 at the University of California. The first implementation, called LANSO, carries out a simple, single vector Lanczos iteration. The second implementation, called BLANSO, carries out block Lanczos iteration. The objective of this study was to modify these implementations as needed in order to adapt them to the basic eigenanalysis system BES used at Lockheed Missiles and Space Company for general eigenanalysis of large structural models. The resulting computer programs were then to be used to calculate eigenpairs for several structural engineering models in order to evaluate LANSO and BLANSO.

Both LANSO and BLANSO were adapted to BES and test studies were carried out on both VAX 11/780 and Cray 1S computers. The tests on the VAX computer were used primarily to determine, develop and check the required modifications of the various codes. The test problems used on the VAX were relatively small (under 500 equations) and are not discussed extensively in this report. All calculations in this study were done in single precision.

1. CODE MODIFICATIONS.

The basic eigenanalysis system BES is a collection of programs and subroutines that interface with several structural analysis programs for modal and buckling analysis. The interface is through a general global database GAL as shown in Fig. 1. Besides the

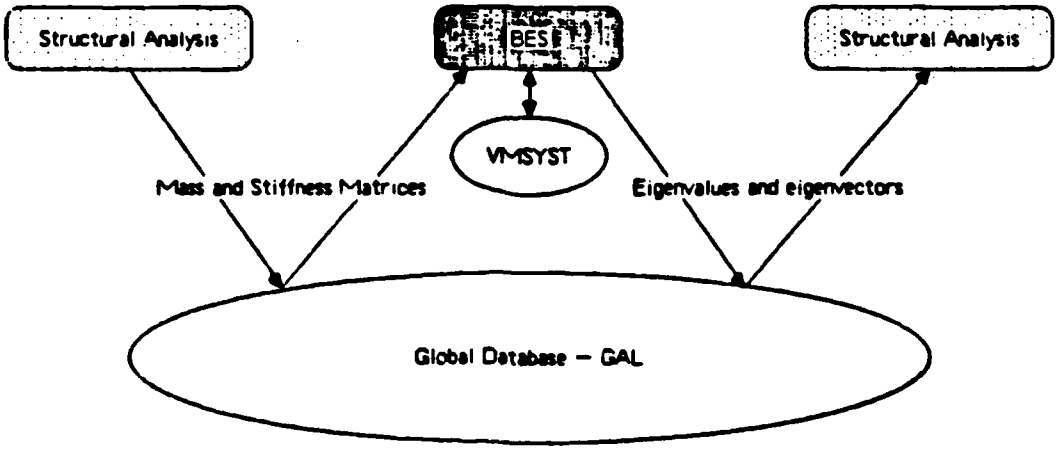


Figure 1. BES Interface System

global database, BES also uses a local working database VMSYST which is based on a virtual memory concept. The fundamental eigen-analysis algorithm used in BES is a block Lanczos implementation originally developed by Prof. David Scott of the University of Texas. This implementation is combined with a spectral shifting algorithm developed by Paul Jensen of Lockheed.

For the present study, BES was divided into three program modules: (1) Root code, (2) Kernel code, and (3) Interface code, as shown in Fig. 2. The largest module, the root code, contains all

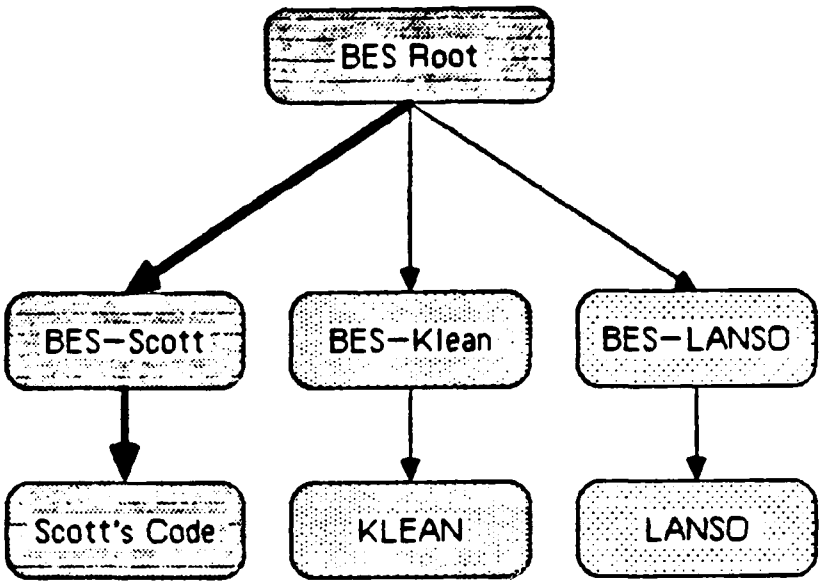


Fig. 2. Partition of BES for Adaptation

utility software for data management, sparse matrix processing, spectral shift processes, error bound calculations and miscellaneous processes not directly associated with Lanczos iteration. The kernel code contains all software developed specifically for Lanczos iteration and the interface code provides the interface between the root code and the kernel code. With this arrangement, a new kernel and interface could readily be constructed for the simple Lanczos program LANSO from the University of California. In addition, a new block Lanczos implementation called KLEAN (Kernel for Lanczos Eigen Analysis) was also adapted to BES. KLEAN is a modification of the BLANSO code developed by Dr. Nour-Omid for this study.

1.1 KERNEL MODIFICATIONS.

A fundamental limitation of both LANSO and BLANSO is the assumption that all of the calculated eigenvectors can be held in

RAM (random access main storage.) This assumption restricts the size of problems that can be solved and can result in unnecessarily early termination due to exhausted RAM space. It can also adversly affect analysis costs levied by some computer billing algorithms that penalize excessive use of RAM.

Elimination of this restriction from LANSO would be rather involved and was not attempted in this effort. The difficulty stems from the use of selective orthogonalization, which requires periodic determination of eigenvectors (Ritz vectors) during the iteration. BLANSO, on the other hand, uses partial reorthogonlization on the Lanczos vectors, and calculates the eigenvectors only once at the end. Modifications included elim- ination of the final eigenvectors calculation, relegating this post iteration task to utility programs already available in BES. The resulting code, called KLEAN, produces only eigenvalues with error bounds, eigenvectors of the reduced (projected) problem, and Lanczos vectors.

Even with these modifications, the RAM requirements of KLEAN are considerably greater than those of Scott's code. A major contributor to this problem is the desire to have space for all of the reduced eigenvectors. This requirement can readily be reduced to sufficient space for the maximum number of solution vectors requested for a given analysis. This reduction, which would require a different solver for the reduced problem, was not implemented in this study.

BES is designed to determine eigenvalues in a specified section of the problem spectrum. This imposes additional requirements on the termination criteria used by LANSO and KLEAN. The iteration should terminate when one or more of the following conditions are satisfied:

1. The number of Lanczos steps reaches a specified limit
2. The number of converged eigenvectors reaches a specified limit
3. All of the eigenpairs with eigenvalues in the specified range have converged.

In cases 1. and 2., the "completed section" of the eigenvalue spectum for which all eigenpairs have converged should be returned. This section, of course, will be a subsection of the one specified initially.

There is no fool-proof, practical method for specifying the completed section or determining that case 3 has been satisfied. Fortunately, the method does not have to be fool-proof because BES has other checks, based on spectral shifting, that insure that the requested eigenpairs are found. The cost for failure to terminate the iteration properly according to the above specifications is extra computing time. Therefore, we need an efficient termination

method that works most of the time.

The natural order for eigenpairs to converge in BES, using either Lanczos or subspace iteration, is to start near the center of the specified section and work outward toward the ends. We currently terminate iteration under criterion 3 above when two eigenpairs outside of the specified section have converged. This heuristic is also used for specifying the completed subsection under termination conditions 1 and 2. The ends of the completed section are the averages of the extreme pairs of converged eigenvalues.

There is a problem when an extreme eigenvalue (near an end of the specified section) is a multiple root. It is not generally cost effective to terminate an iteration (e.g. due to condition 2 above) before all eigenvectors of that eigenvalue are determined. In its generic form, BES utilizes both a "hard" limit, as in 2, and a "soft" limit which occasionally is exceeded when appropriate. This innovation was not incorporated in LANSO or KLEAN.

1.2 INTERFACE MODIFICATIONS.

Most of the interface code development was routine and tedious in nature. The primary interface routines that are called by the kernel code are matrix utilities and vector storage and retrieval routines. These are simple relative to the routines that establish operating parmeter values, and monitor and control iteration progress. For small problems this is not difficult but for large problems, certain trade-off decisions between available RAM and working storage allocation must be made that have a substantial effect on the results. The interface code developed for LANSO and KLEAN in this study was adequate for this work but should be extended for production analysis.

Additional interface software required for this work is concerned with postprocessing the results from the kernel code. The exact nature of this code varies, depending upon the nature of the kernel code used. The most dependable postprocessing scheme appears to be one step of inverse iteration followed by an astutely ordered, modified Gram-Schmidt orthonormalization. This, however, is rather expensive. An alternative postprocessing scheme is derived from

$$K^{-1} M Q_j = Q_j T_j + Q_{j+1} B_{j+1} E_j^T \qquad (1)$$

which is the jth step of the Lanczos interation process. Post-multiplying (1) by the matrix S of converged eigenvectors for T yields a simple expression for one step of inverse iteration applied to the Ritz vectors

$$Y = Q S_j$$

Tests in which the inverse iteration post-processing step was carried out using the right side of (1) indicate that this is an effective approach.

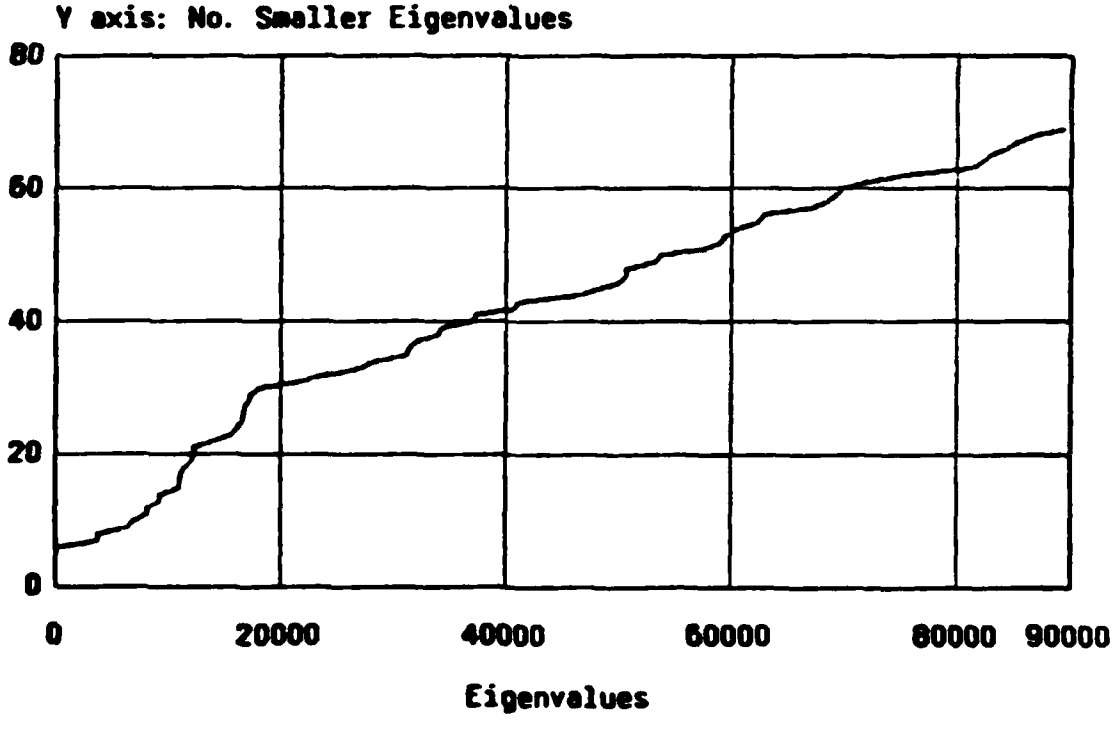
2. TEST PROBLEMS

Three problems from typical engineering studies carried out at Lockheed Missiles and Space Company were used for this study. The nature of the structures represented in these problems is not material and we view them simply as sparse matrices for which eigenvalues are required. All of them are of the form

$$Kx = Mx\lambda$$

where K and M are non-negative definite and M is diagonal with rank ranging from less than half to about two-thirds of its size. All tests on these problems were run on a Cray 1S computer in single precision. The distribution of the eigenvalues over a small portion of a generalized spectrum is illustrated in Fig. 3. The ordinate

Figure 3. Spectral Distribution for Problem 3



for each eigenvalue indicates the number of smaller eigenvalues that exist. Thus, the sharp rises in the curve in Fig. 3 correspond to multiple or clustered eigenvalues. The results of the numerical tests are summarized in Table 1.

Unfortunately, no results for the large problems using LANSO were obtained. The code would function very well for 20 to 30 iterations and then fail. The problem turned out to be that more eigenvalues would converge simultaneously than the code was prepared to handle. This determination was made so late in the study that it was not possible to include test results in this report.

Note that the CPU time for problem 3 (indicated in Table 1b by *) is not available because it represents a subsection analyzed as a part of a larger problem. The subsections for different algorithms were identical, however, making the comparisons valid.

The relatively small rank of these problems appears to be typical of engineering structural analysis. This factor tends to favor a symmetric formulation of the Lanczos iteration matrix as used in Scott's code, but the recorded CPU times do not indicate a significant benefit. In fact, the iteration time per Lanczos step (It/L) tends to be smaller for the new algorithm. The factorization and solve times for the two methods, of course, are about the same because they use the same sparse matrix processing utilities. This result is probably due to the relatively large amount of reorthogonalization done in Scott's code.

Table 1b also includes some results comparing the automatic spectral shifting algorithm in BES with a hand selected shift strategy. The hand selection of shift points requires that the solution be already known. The results are encouraging from the standpoint that the hand selected set reduced the total processing time by only 21%. However, there is presently no basis to claim that the automatic shifting algorithm used is optimal.

The new algorithm KLEAN tends to iterate longer for each set of eigenvalues. The stopping criteria appears to be too strict and probably can be relaxed. This and the question of when and where to do a spectral shift remain to be settled in future research.

Problem 2 and some smaller problems were run using both the standard inverse iteration post-processing step and technique described in Sec. 1.2 with the KLEAN kernel. The accuracy of the results were similar and the savings in overhead time for problem 2 was 3 seconds or 6%.

Table 1a. Legend for Table 1b Column Headings

P	- Problem: 1, 2 or 3
M	- Method: S - Scott's code, K - KLEAN, L - LANSO
Size	- Number of degrees of freedom in the problem
Rank	- Rank of the M matrix
Fac.	- Time in seconds to factor (K - (shift)*M)
E	- Number of converged eigenpairs found
Iter.	- Time in seconds required for the Lanczos iteration
L	- Number of Lanczos steps taken
B	- Block size used for the Lanczos iteration
It/L	- Iteration time per Lanczos step (Iter./L)
BL/E	- Matrix-vector multiplies per converged eigenpair (B*L/E)
Total	- Total CPU seconds required for the analysis
OH	- Percentage of the total time used for processing other than factoring or iterating (Overhead)

Table 1b. Summary of Numerical Results

P	M	Size	Rank	Fac.	E	Iter.	L	B	It/L	BL/E	Total	OH
1	S	3116	2236	7.86	25	10.58	16	3	0.66	1.92	29.54	37
1	K	3116	2236	7.87	25	8.67	18	3	0.48	2.16	23.65	30
2	S	4338	2217	16.72	18	15.41	16	3	0.96	2.67	42.77	25
2	K	4338	2217	16.64	18	17.42	19	3	0.92	3.17	43.70	22
2	S	4338	2217	16.74	9	9.65	11	3	0.88	3.67	35.21	25
2	K	4338	2217	16.70	9	12.65	14	3	0.90	4.67	34.84	16
3	S	4614	1927	10.40	16	8.92	15	3	0.59	2.81	*	
3	K	4614	1927	10.33	16	9.75	16	3	0.61	3.00	*	
3	S	4614	1927	10.42	13	9.74	16	3	0.61	3.69	*	
3	K	4614	1927	10.31	13	13.69	22	3	0.62	5.08	*	
3	S	4614	1927	10.42	19	41.11	49	3	0.84	7.74	179.60	22
				10.40	17	16.29	23	3	0.71	4.05		
	Auto. spectral			10.39	13	12.60	21	3	0.60	4.84		
	shifting			10.42	17	10.89	18	3	0.60	3.18		
				10.37	5	7.04	13	3	0.54	7.80		
3	S	4614	1927	10.39	14	10.18	17	3	0.60	3.64	141.00	26
				10.40	16	8.92	15	3	0.59	2.81		
	Hand specified			10.42	13	9.74	16	3	0.61	3.69		
	spectral shift			10.40	13	9.74	16	3	0.61	3.69		
				10.41	14	11.74	19	3	0.62	4.07		

Block Tridiagonal Matrix Analysis for the Lanczos Algorithm.

1. Background.

We are interested in computing some of the eigenpairs of the generalised eigenproblem

$$Kx = \lambda Mx \qquad (*)$$

Here K and M are real and symmetric matrices, of order n. Usually the matrices are large and sparse. M is at least positive semidefinite. The eigenvalues of interest usually lie in some interval [a, b], specified by the user.

The problem may be transformed into

$$(K-\mu M)M^{-1}x = (\lambda-\mu)^{-1}x \qquad (**)$$

provided the shift, μ, has been chosen so that $K-\mu M$ is nonsingular. One efficient way to solve our problem is to apply a Lanczos algorithm to the shifted and inverted problem (**). Let us for a moment consider the simple Lanczos algorithm (not a block version). The Lanczos algorithm will produce vectors $v_i \in \mathbb{R}^n$ and real numbers α_i, β_i (the β_i are positive), such that in exact arithmetic :

$$V_j = (v_1, \ldots, v_j), \quad \text{with} \quad V_j^T M V_j = I$$

and

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & & & \\ & & \ddots & & \\ & & & \alpha_{j-1} & \beta_{j-1} \\ & & & \beta_{j-1} & \alpha_j \end{bmatrix}$$

such that

$$(K-\mu M)^{-1}MV_j = V_j T_j + \beta_j v_{j+1} e_j^T$$

(Here e_j is column number j in the identity matrix.)

Usually $j \ll n$, typically n = 1000–5000 with j = 50–100. Now, some of the eigenvalues of T_j will be close to some eigenvalues of problem (**). Usually the end-eigenvalues of T_j have the best approximation properties. If (s, ν) is an eigenpair of T_j, then

$$\| (K-\mu M)^{-1}MV_j s - \nu V_j s \|_M = \beta_j |s_j|$$

(Here $\|u\|_M = (u^T M u)^{1/2}$.)

If $\beta_j |s_j|$ is small, then $(\nu, V_j s)$ will be a good approximation to an eigenpair of (**). Usually the product is small because $|s_j|$ is small, and not because β_j is. Since we are really interested in problem (*), we can transform the ν-eigenvalues back to the corresponding λ-eigenvalues. It is easy to see that the transformation is given by $\mu + 1/\nu$. We said above, that extreme eigenvalues of T_j have the best approximation properties, and so they correspond to λ-eigenvalues relatively close to μ.

To know when to stop a Lanczos run (j is not usually determined beforehand, but should be computed during the iteration, taking into account the rate of convergence, cost of another Lanczos step, number of requested eigenpairs etc.), it is essential to know which eigenpairs of T_j are good approximations. We would, thus, like to have an efficient and reliable algorithm that computes the interesting (ν, s)-pairs in each iteration of the Lanczos algorithm.

Such an algorithm should not start from scratch for each T_j, but it is desirable to update the spectral information computed for T_{j-1}. The following figure illustrates this idea:

I = interesting eigenvalue (i.e. good approximation)
\+ = uninteresting eigenvalue
$\lambda(T)$ = eigenvalues of T

--]--]---]-----+--+--+-+++--+---+---+-----]-]-------]-- $\lambda(T_{j-1})$

-]--]---]---^]---+--+--+-+++--+---+---+----]-]-------]- $\lambda(T_j)$

In this example the three extreme eigenvalues, at each end, have not changed much, and should therefore be easy to update. There is one new eigenvalue (there must of course be one new eigenvalue, since the order of T has increased by one), which in this case is an interesting one (marked ^). The new eigenvalue need not, of course, be interesting. If it comes in the middle, for example, it is not likely a good approximation.

This problem has been attacked by B. Parlett and B. Nour-Omid, resulting in an algorithm, analyze T.

2. The Block Case.

B. Parlett and myself have now looked into the similar problem arising from a run of a *block* Lanczos algorithm. So instead of producing vectors v_i and numbers α_i, β_i, the block Lanczos algorithm will produce blocks of vectors $V_i \in \mathbb{R}^{n \times p}$, and square matrices A_i, $B_i \in \mathbb{R}^{p \times p}$ (A_i is symmetric and B_i is lower triangular). Let p be a fixed integer, called the *block-size*. It is supplied by the user, or is set by the program. Typical values are 3-7. Note that $p = 1$ gives us the simple Lanczos described above. Instead of the tridiagonal matrix, in the simple Lanczos, a block tridiagonal matrix

$$T_j = \begin{pmatrix} A_1 & B_1 & & & \\ B_1^T & A_2 & & & \\ & & \ddots & & \\ & & & A_{j-1} & B_{j-1} \\ & & & B_{j-1}^T & A_j \end{pmatrix}$$

will result. (We use the same notation for this new matrix. Note that the order of T_j is jp.) Since the B_i are lower triangular, T_j is a bandmatrix having half bandwidth p. A residual bound can be computed as the norm of the product of the last block and the bottom segment of the eigenvector (this corresponds to $\beta_j |s_j|$ when $p = 1$).

We are looking for a new algorithm that solves essentially the same problem as in the simple case, i.e. it should monitor the behaviour of the extreme eigenvalues of T_j, now being a block tridiagonal matrix. As before we have a sequence of matrices T_1, T_2, ..., but for each block Lanczos iteration, the order of T_j increases by p.

3. Some Reasons Why the Block Case is Harder to Deal With

This change, from $p = 1$ to $p > 1$, may not seem very dramatic, but it gives rise to several new difficulties, of which some will be presented below.

If $p = 1$ (the β_i are assumed to be positive, i.e. T_j is unreduced (the Lanczos algorithm would have been stopped if a very small β_j had occurred)) T_j has *distinct* eigenvalues. T_j may, of course, have very close eigenvalues (even if not any β_j is small, compare Wilkinson's famous example (The Algebraic Eigenvalue Problem, page 309)). When $p > 1$, T_j may have truly multiple eigenvalues. This is, in fact, one reason why the block algorithm is superior for handling multiple eigenvalues in the original problem (*).

If $p = 1$ the eigenvalues of T_j and T_{j+1} strictly interlace each other. This property gives rise to several useful results:

(1) It is easy to find the new eigenvalues of T_{j+1}, since there can only be one between two eigenvalues of T_j (and we know most of the interesting eigenvalues of T_j).

(2) We can estimate gaps between eigenvalues (which implies that we can use more powerful error bounds).

These two properties fail when $p > 1$. Going from T_j to T_{j+1} we may have several new eigenvalues between two of T_j. In fact, using Cauchy's interlace theorem, we can only say that between two eigenvalues of T_j there may be p eigenvalues of T_{j+1}. These new ones can be grouped together, coincide with eigenvalues of T_j (even though the B_i are nonsingular). Since p may be rather large, the spectrum may have undergone large changes between two consecutive calls of the analyse routine. Consider the following, little example.

Example.

Assume $p = 2$, and let $A_1 = A_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, and let $B_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$. Then T_1 has eigenvalues -1 and 1, and T_2 has eigenvalues $-\sqrt{2}$, $-\sqrt{2}$, $\sqrt{2}$, $\sqrt{2}$.

One way to simulate this interlace property, for the bandmatrix, is the following: Let T_{j+1}^k denote the leading principal submatrix, of order k, of T_{j+1}. Then the eigenvalues of T_j and T_{j+1}^{p+1} must interlace each other (though not necessarily strictly). The same is true for all pairs T_{j+1}^{p+k}, T_{j+1}^{p+k+1}, $1 \le k \le p-1$. It would be possible to use this idea when trying to locate eigenvalues.

Yet another problem is the fact that p is not fixed (p depends on the problem

(*)). If the problem has eigenvalues of high multiplicity, or if I/O is expensive, it may pay to use a larger value on p. A good algorithm should try to take into account varying values of p. To be specific, suppose that inverse iteration is used in the analyse routine. We could consider an inverse iteration of the form:

Given a starting vector y_1 and first shift σ

For k = 1, 2, 3 ... until convergence do
{ factor $T_j - \sigma I$

$b = y_k$
For i = 1 to m_k do
{ solve $(T_j - \sigma I)z = b$
$b = z / \|z\|$ }
$\rho = \rho(b)$ ($\rho(b)$ is the Rayleigh quotient) }

$y_{k+1} = b$
$\sigma = \rho$ }

Taking $m_k = 1$, for all k, gives the standard Rayleigh quotient iteration. It may be advantageous to vary m_k. If, for example, y_1 is not too close to the requested eigenvector, the first shift may be rather poor, and it may give faster overall convergence, if the next shift is refined by using several solves.

Example.

In this example some output, from an algorithm like the one above, is presented. $T_j = diag(0, 1, 2, 2, 3, 6.5, 7, 8, 9, 10)$. The first shift was 3.6 (a rather poor shift). The starting vector was:

$y_1 =$

3.7796447e-01	3.7796447e-01	3.7796447e-09
3.7796447e-09	3.7796447e-01	3.7796447e-01
3.7796447e-09	3.7796447e-01	3.7796447e-01
3.7796447e-01		

One should interpret this y_1 in the following way: 2, 2, and 7 are known eigenvalues, and we have orthogonalised the starting vector against the known eigenvectors. We have the same components in the other eigenvectors.

What follows below is a printout, for a sequence of runs, where m_k was 1, 2, 3, etc. Note that m_k did not vary with k. The first column indicates what factorisation is being used. Repeated values means that we make several solves, using the same factorisation. The second column contains $1/\|z\|$. The third is the Rayleigh quotient formed after <u>each</u> solve. The last column contains rho-lambda, the difference between the Rayleigh quotient and the requested eigenvalue (which is 3). The termination criterion was abs(rho-lambda) ≤ 1e-16 (in this special case we knew λ).

```
      1/ ||z||          ρ                 error
1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
2   1.7475323e-01   2.9997320e+00   2.6801401e-04
3   2.6808211e-04   3.0000000e+00   9.9601993e-12
4   9.9601993e-12   3.0000000e+00   0.      e+00

#solves / step =  1
factorisations =  4 (total number of factorisations)
solves         =  4 (total number of solves)
```

==========

```
      1/ ||z||          ρ                 error
1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
2   1.6056911e-03   3.0000000e+00   2.6730501e-09
2   1.6008833e-03   3.0000000e+00   2.2204460e-15
3   2.2204460e-15   3.0000000e+00   0.      e+00

#solves / step =  2
factorisations =  3
solves         =  5
```

==========

```
      1/ ||z||          ρ                 error
1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
1   6.0172374e-01   2.9999564e+00   4.3632258e-05
2   4.3637928e-05   3.0000000e+00   1.1146639e-13
2   4.3632258e-05   3.0000000e+00   0.      e+00

#solves / step =  3
factorisations =  2
solves         =  5
```

==========

```
      1/ ||z||          ρ                 error
1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
1   6.0172374e-01   2.9999564e+00   4.3632258e-05
1   6.0007432e-01   2.9999947e+00   5.3417704e-06
2   5.3418028e-06   3.0000000e+00   1.1102230e-16
2   5.3417704e-06   3.0000000e+00   0.      e+00

#solves / step =  4
factorisations =  2
solves         =  8
```

```
==========================================================

       1/||z||          ρ                 error
 1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
 1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
 1   6.0172374e-01   2.9999564e+00   4.3632258e-05
 1   6.0007432e-01   2.9999947e+00   5.3417704e-06
 1   6.0000347e-01   2.9999996e+00   3.9003350e-07
 2   3.9003362e-07   3.0000000e+00   0.      e+00

 #solves / step =  5
 factorisations =  2
 solves         =  6


==========================================================

       1/||z||          ρ                 error
 1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
 1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
 1   6.0172374e-01   2.9999564e+00   4.3632258e-05
 1   6.0007432e-01   2.9999947e+00   5.3417704e-06
 1   6.0000347e-01   2.9999996e+00   3.9003350e-07
 1   6.0000017e-01   3.0000000e+00   2.5234880e-08
 2   2.5234881e-08   3.0000000e+00   0.      e+00

 #solves / step =  6
 factorisations =  2
 solves         =  7


==========================================================

       1/||z||          ρ                 error
 1   1.4716494e+00   3.1620465e+00  -1.6204650e-01
 1   6.4527501e-01   3.0016009e+00  -1.6008833e-03
 1   6.0172374e-01   2.9999564e+00   4.3632258e-05
 1   6.0007432e-01   2.9999947e+00   5.3417704e-06
 1   6.0000347e-01   2.9999996e+00   3.9003350e-07
 1   6.0000017e-01   3.0000000e+00   2.5234880e-08
 1   6.0000001e-01   3.0000000e+00   1.5419184e-09
 2   1.5419184e-09   3.0000000e+00   0.      e+00

 #solves / step =  7
 factorisations =  2
 solves         =  8

==========================================================
```

Which of these runs is optimal? Well, it depends on the cost ratio between factor and solve. If n is the order of T_j, i.e. $n = jp$, then the costs (or rather, the operation counts) are:

$$\text{factor} \quad \frac{n^2 p}{2} - \frac{p^3}{3} + \frac{3}{2}(np - p^2) \approx \frac{np}{2}(p+3)$$

$$\text{solve} \quad n(2p+1) - p^2 + 3n \approx 2n(p+2)$$

The extra $3n$-term in solve comes from normalising the solution, and computing the Rayleigh quotient. (There are other ways to measure convergence, not using the Rayleigh quotient, but we will not discuss that here.) Below follows a table over the ratio, factor/solve, for varying p.

p	factor / solve
1	0.3333
2	0.6250
3	0.9000
4	1.1667
5	1.4286
6	1.6875
7	1.9444
8	2.2000
9	2.4545
10	2.7083
11	2.9615
12	3.2143
13	3.4667
14	3.7188
15	3.9706
20	5.2273
25	6.4815
30	7.7344

For large values of p, the quotient behaves like $p/4$.

Depending on p, different m_k will be optimal. If, for example, factor is more expensive than solve, 3 solves/factor is optimal (since it minimises the total cost in this example). If $p = 1$, usual RQI is the optimal choice.

I have done some tests letting m_k vary during one run, but the results are too preliminary to be included here.

3.1. Algorithm 1.

The first algorithm was a try to generalise analyse T (for $p = 1$) to $p > 1$. To avoid the wealth of detail necessary to specify the algorithm in an algorithmic language, we will only present the basic strategy using a few figures.

I = eigenvalue converged to working accuracy
* = eigenvalue on the way to converge
+ = unconverged eigenvalue (these eigenvalues may move around quite a bit, and they need not settle down)
$\lambda(T)$ = eigenvalues of T

--I--I---*-----+--+--+-+++--+---+---+----*-I-------I- $\lambda(T_{j-1})$

--I--I--^---*---+--+--+-+++--+---+--~-----II-------I-- $\lambda(T_j)$

We keep information about the eigenvalues (and corresponding eigenvectors) marked I and *, but *no* information about the + eigenvalues. This is a reasonable approach, since we expect the end-eigenvalues to be good approximations. The ones in the middle are quite useless for our purposes. We also keep information about the bounds for the eigenvalues (this is what gives the I,*-classification).

Starting at the left end of the spectrum of T_{j-1} (we do not know the spectrum of T_j. That is what we would like to find out.), march through the I and * eigenvalues, and update (i.e. refine the eigenvalue and eigenvector using inverse iteration) the *-eigenvalues. We do not have to refine the I-eigenvalues, since they are fully converged. In this process some *-eigenvalues can become I-eigenvalues (but not the other way). In the example, the one marked ^ has undergone this transformation. Having finished with the left side, we do the same thing on the right side (but in the reverse order). To be able to continue this process, we must probe into the unknown region in the middle of the spectrum, since, if do not get any new *-candidates, we would end up with a fixed number of I-eigenvalues. The eigenvalue marked ~ is such a new candidate. The actual algorithm starts at the innermost * or I-eigenvalue (at each side) and adds new eigenvalues as long as they qualify as *-eigenvalues.

3.1.1. Intruders.

There is one important case which we have not treated so far. It is illustrated in the following figure:

... --*-----*-------*-+-------+--+-- ..... $\lambda(T_{j-1})$

... -*-----*-----*---*---+------+--+-- ..... $\lambda(T_j)$

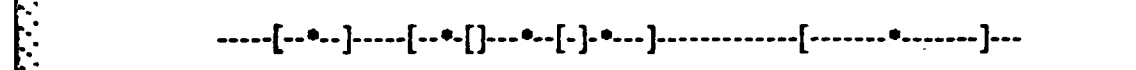
In this case we have a new eigenvalue (marked ^) between two known ones. We would detect this situation using the eigenvalue count we get from the factorisation (when doing the inverse iteration). This new eigenvalue must be located (we only know an interval in which it lies), and we use a combination of bisection and inverse iteration.

One algorithmic detail: To get a clearer code, we represent the eigenvalues at the right end, as the left end eigenvalues of the matrix $-T_j$. So, we deal with two left ends.

Now, when $p > 1$, we can get clusters of eigenvalues, multiple eigenvalues, and several intruders between a new pair. I did write a FORTRAN program following the principles above. It worked quite well for distinct eigenvalues, but not so for clusters. We have not given up this algorithm yet, but several problems need to be solved to get a working program. Due to these problems we considered:

3.2. Algorithm 2.

Since we are dealing with clusters it is reasonable to used a method which determines higher order subspaces, instead of simple eigenvectors. One such method is subspace iteration. In this second algorithm (which I will not discuss), we replaced simple inverse iteration by subspace iteration. A cluster would be defined by having overlapping bounds, as in the following figure, where the two end-eigenvalues are isolated, and the three in the middle form a cluster.



[...] marks the interval $\lambda \pm bound$.

One worry at this point, was that the code was becoming quite long. In particular we had problems with the part finding intruders. It is, in fact, the completely general problem: Given an interval [a, b] compute a known number of eigenvalues in the interval. This lead us to look at yet another algorithm :

3.3. Algorithm 3.

This is the simplest algorithm we have considered, and it is quite expensive. The idea is to first reduce T_j to tridiagonal form. This can be made using orthogonal rotations. (Starting with the first row, zero the right-most element, this will give rise to a new element, outside the band. This new element is chased over the edge with a sequence of rotations. Then zero the next element in the first row etc. Continue this process row-wise.) It is quite an expensive algorithm, the operation count is $\leq \frac{1}{2}n^2(p-1)(4+13/2p)$, where $n = jp$ as usual. (Note that it is a n^2p process, factor and solve are np^2 and np processes.)

Having made the reduction, we compute *all* the eigenvalues of the tridiagonal matrix. This is considerable cheaper than the initial reduction (at least on computers with standard architecture. According to B. Nour-Omid, Householder reduction on a full matrix, on a CRAY, is comparable in time to finding the eigenvalues of the tridiagonal). We would save the eigenvalues from the previous step (so $\lambda(T_{j-1})$ and $\lambda(T_j)$, are available). Now we would make a comparison between the two sets of eigenvalues. If an eigenvalue of T_j is close to any of T_{j-1}, we would compute the corresponding eigenvector using inverse iteration. If the eigenpair is a good one (we compute a bound using B_j and the eigenvector) it is saved, otherwise it is discarded. Sometimes we would already have an approximation of the eigenpair, and it would suffice to update the eigenvector. To make this strategy work well in presence of clusters and multiple eigenvalues, we would need a carefully coded inverse iteration. (It could, for example, orthogonalise the starting vector against known eigenvectors (having eigenvalues close by), and it might check the orthogonality of the resulting eigenvector to the previously computed.)

4. Other Alternatives.

We have also considered some other alternatives. One that does not work (unfortunately) is the following: Given T_j in tridiagonal form (and the transformation matrix giving the reduction), perform a few operations to get T_{j+1} (given B_{j+1} and A_{j+1}) in tridiagonal form. The cost of doing this seems to be equal to starting from scratch with T_{j+1}.

Another alternative, not very cheap, is to perform the reduction in each step (from scratch), and then feed the resulting tridiagonal (starting at step $jp+1$) to analyze T. The main advantage is the extreme simplicity, essentially two subroutine calls.

Other solutions would be to use the Lanczos algorithm, and preferably the block algorithm, to solve the subproblem, but it tends to give a lot of code (or demand recursion).

We are presently working on ways to change algorithm No. 1 to cope with clusters and other problems, and we hope that these efforts will result in a fast and reliable algorithm.

END

FILMED

11-85

DTIC