UC SESM 71-20

STRUCTURES AND MATERIALS RESEARCH
DEPARTMENT OF CIVIL ENGINEERING

SOLUTION METHODS FOR LARGE GENERALIZED EIGENVALUE PROBLEMS IN STRUCTURAL ENGINEERING

by

KLAUS-JÜRGEN BATHE

NOVEMBER 1971

STRUCTURAL ENGINEERING LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY CALIFORNIA

Structures and Materials Research
Department of Civil Engineering
Division of Structural Engineering
and Structural Mechanics

SOLUTION METHODS FOR LARGE GENERALIZED EIGENVALUE PROBLEMS IN STRUCTURAL ENGINEERING

by

Klaus-Jürgen Bathe

Dissertation Committee

E. L. Wilson
R. W. Clough
Professors of Civil Engineering
University of California, Berkeley

B. N. Parlett
Professor of Computer Science
University of California, Berkeley

Structural Engineering Laboratory
University of California
Berkeley, California

November 1971

ABSTRACT

In dynamic and buckling analysis of structures a few eigenvalues and associated eigenvectors may be required in the solution of the generalized eigenvalue problem $Av = \lambda Bv$, where A is positive definite. Currently, when the order of the matrices is very large, approximate solution techniques are used. The aim in this research was the development of efficient computer programs which can find the required eigenvalues and corresponding eigenvectors of large systems to the desired accuracy.

First the dynamic and buckling problems are presented in which the generalized eigenvalue problem arises. In this discussion the special properties of the operators A and B in structural analysis are identified and the eigenproblem solution requirements are stated. Also, approximate solution techniques which are commonly used are critically reviewed. In particular, it is pointed out, that in these analyses we do not know about the accuracy of the eigenvalue approximations obtained, and that an approximation to an important eigenvalue can be missed altogether.

The numerical details of two solution algorithms are then developed. The first technique is a determinant search method, in which triangular factorization and vector inverse iteration is combined in a very efficient manner. The second algorithm is a subspace iteration, which is more economical when the bandwidth of the system is large. Both techniques solve the generalized eigenvalue problem without a transformation to the standard form. There are no difficulties when B is banded or diagonal non-

negative definite. Also eigenvalues and corresponding vectors need not be found to high precision for numerical stability.

Operation counts are evaluated to develop maximum efficiency in the iterations and to give cost estimates of using the algorithms. The study of the number of operations and of numerical aspects is used in the development of a program for the practical case B diagonal and non-negative definite when A has any order and bandwidth.

Various practical example analyses including the analysis of dam, a plane frame and a three-dimensional building frame are presented to show the capabilities and convergence characteristics of the solution techniques.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS

This research was part of my graduate study for the Ph.D. degree in Engineering at the University of California, Berkeley.

I would like to express my deepest gratitude to Professor E. L. Wilson and to Professors B. N. Parlett and R. W. Clough for their helpful guidance and support. Thanks are also due to Professor W. Kahan for many stimulating discussions.

I am very thankful to the University for providing me with the Popert Wm.H. and Helena I.S. fellowship during the last two years, and to Professor A. C. Scordelis for supporting me in the beginning of my graduate studies at Berkeley.

LIST OF SYMBOLS

All symbols are defined in the text, and those not frequently used may have different meaning in different parts of the thesis. Only those which do not change meaning and are often used are listed below.

A = band matrix

B = band or diagonal matrix

$\tilde{L}$ = Cholesky factor of B

n = order of A and B

m_A = half bandwidth of A ; bandwidth = $2m_A+1$

m_B = half bandwidth of B ; bandwidth = $2m_B+1$

λ_i = eigenvalues of $Av = \lambda Bv$

Λ = diagonal array of eigenvalues λ_i

v_i = eigenvectors of $Av = \lambda Bv$

V = matrix storing the eigenvectors v_i

x_k = iteration vector

X_k = matrix storing iteration vectors

Λ_k = diagonal matrix storing eigenvalue approximations

A_k = projection of A

B_k = projection of B

μ_k = shift

$p(\mu)$ = characteristic polynomial

I = identity matrix

e_i = i'th column of I

1. INTRODUCTION

In this dissertation we consider the generalized eigenvalue problem

$$Av = \lambda Bv \tag{1.1}$$

where A and B are symmetric matrices of order n and half bandwidth m_A and m_B, respectively. This equation arises in dynamic and buckling analysis of structures. At least one of the matrices is positive definite. There are n real eigenvalues λ_i and corresponding orthonormalized eigenvectors which we order as

$$\lambda_1 \leq \lambda_2 \leq \lambda_3 \ldots \leq \lambda_n$$

$$v_1 \; ; \; v_2 \; ; \; v_3 \; ; \; \ldots \; ; \; v_n$$

During recent years the ability to perform structural analyses has improved significantly. The finite element method used on a digital computer can allow earthquake or buckling analyses of large and very complex systems [1]. In these analyses the solution of Eq. (1.1) is required for only a few eigenvalues and vectors. Procedures which solve for all eigenvalues and which do not take advantage of the banding characteristics in A and B are at least inefficient and regarding storage may well be impossible to use.

Currently, when the order of the eigenvalue problem is very large and we only require a few eigenvalues approximate methods are used for solution [2].

The object of this research was the development of efficient programs which solve for a few required eigenvalues and associated

vectors in Eq. (1.1). The matrices A and B shall be allowed to be of different forms of practical significance.

In the thesis, first the buckling and dynamic problems in which Eq. (1.1) arises are presented. This leads to a description of the operators A and B in each case and a proper statement of the large eigenvalue problems to be considered. Then the theory and algorithms developed for solution are given. Whenever possible, the theory is presented to obtain a good physical understanding. Finally, example analyses are given to show the convergence characteristics of the programs.

Basically two different solution techniques have been developed. A determinant search algorithm was implemented which combines triangular factorization with vector inverse iteration in a very efficient manner. The program is most efficient in the analysis of systems with small bandwidth. The other scheme is a subspace iteration which is economically used on systems with larger bandwidth. Included in the thesis is a routine for the practical case B diagonal and non-negative definite to allow the solution for any matrix size and bandwidth. In the determinant search algorithm and the subspace iteration, Eq. (1.1) is solved directly without transforming to the standard eigenvalue problem. The programs are given in Appendix II. They are written in Fortran IV and have been tested on the CDC 6400 at the University of California at Berkeley.

In my opinion the solution of eigenvalue problems is most fascinating for its theory and the large variety of practical

numerical problems. For those readers who are not very familiar with the problem some background in the theory and efficient solution methods for the problem $Av = \lambda v$ with A being small are given in Appendix I.

2. THE LARGE GENERALIZED EIGENVALUE PROBLEM IN STRUCTURAL ANALYSIS

2.1 Introduction

Before the different solution procedures to Eq. (1.1) are presented, we should discuss the cases in dynamic and stability analysis where Eq. (1.1) arises. This way we can observe the different properties which the operators A and B can take and define the large eigenvalue problems that we consider. The approximate solution techniques used in practice will also be presented and critically discussed.

2.2 The Eigenvalue Problem in Dynamic Analysis

The equations of motion for a system of structural elements can be written as

$$M\ddot{u} + C\dot{u} + Ku = P \tag{2.1}$$

where M is the mass, C the damping and K is the stiffness matrix of the system, all of order n. Vectors u and P store displacements and forces, respectively [2]. The matrices M, C and K are obtained in conventional beam analysis of structures, in the two and three dimensional finite element discretization of continua and using other techniques of analysis, such as the finite difference method.

Equation (2.1) is solved by considering first free vibration conditions, where

$$M\ddot{u} + Ku = 0 \tag{2.2}$$

Substituting

$$u = \varphi \sin \omega(t - t_o)$$

we obtain the generalized eigenvalue problem

$$K\varphi = \omega^2 M\varphi \tag{2.3}$$

The n eigenvalues give the natural frequencies of the system and the eigenvectors the corresponding vibration modes. The complete solution to Eq. (2.3) can be written as

$$K\Phi = M\Phi\,\Omega^2 \tag{2.4}$$

where the columns in Φ are the eigenvectors φ_i and $\Omega^2 = \text{diag}\,(\omega_i^2)$, with $\omega_i^2 > 0$ all i.

We now change basis from the physical coordinate basis to the M-orthogonal basis of eigenvectors. Thus Eq. (2.1) becomes

$$\ddot{X} + C^*\dot{X} + \Omega^2 X = \Phi^T P \tag{2.5}$$

where X lists the coordinates in the new basis and $C^* = \Phi^T C \Phi$. If, in practice, C^* is assumed to be diagonal, Eq. (2.5) consists of n decoupled equations, which are readily solvable [2].

The most time consuming step in the analysis is the eigenvalue solution. If the size of the matrices is large, the computer time required to solve for all eigenvalues and vectors is enormous. However, experience has shown that many structures respond to particular types of dynamic loading primarily in a few modes, and that the contribution of the other modes can be neglected. For example, in earthquake response analysis it can be sufficiently accurate to consider only the lowest eigenvalues and corresponding vectors. Naturally, the exact number of modes to be included in an analysis depends on the structure, the loading and the accuracy sought. But provided the required eigenvalues and vectors in Eq. (2.3) can be found with a reasonable computer effort, large dynamic systems can be analyzed.

Consider in more detail the eigenvalue problem in Eq. (2.3). It is of particular importance that in structural analysis both matrices K and M are banded, i.e.

$$k_{ij} = 0 \text{ for } j > i + m_A$$

$$m_{ij} = 0 \text{ for } j > i + m_B$$

where $(2m_A + 1)$ and $(2m_B + 1)$ are the bandwidths of the matrices. Assuming that all rigid body modes have been removed from the system, K is positive definite. If, in a finite element analysis a consistent mass formulation is used, M is also positive definite and $m_B = m_A$. However, experience has shown that a consistent mass formulation is often not necessary and good accuracy can be obtained in a lumped mass analysis. Then M is diagonal with m_{ii} positive or zero.

Because the order of the eigenvalue problem in Eq. (2.3) can be several hundred, approximate techniques have been developed to reduce computational requirements for finding the few lower modes.

If it can be justified to lump all mass at some specific degrees of freedom, we can rewrite Eq. (2.3) as

$$\begin{bmatrix} K_{aa} & K_{ac} \\ K_{ca} & K_{cc} \end{bmatrix} \begin{bmatrix} \varphi_a \\ \varphi_c \end{bmatrix} = \omega^2 \begin{bmatrix} m_a & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \varphi_a \\ \varphi_c \end{bmatrix} \tag{2.6}$$

We can now use static condensation on the φ_c degrees of freedom and obtain the eigenvalue problem

$$K_a \varphi_a = \omega^2 m_a \varphi_a \tag{2.7}$$

where

$$K_a = K_{aa} - K_{ac} K_{cc}^{-1} K_{ca}$$

In practice, K_a can be obtained as follows

$$K_{cc} = LL^T \; ; \; LY = K_{ca} \; ; \; K_a = K_{aa} - Y^T Y$$

Alternatively, we may modify the complete structure stiffness by using Gaussian elimination on the φ_c degrees of freedom. However, a scheme is then needed to take account of the increasing bandwidth of the system.

For solution of φ_c we can use the relation $K_{cc}\,\varphi_c = -K_{ca}\,\varphi_a$ given in Eq. (2.6).

In the analysis we could also solve

$$\begin{bmatrix} K_{aa} & K_{ac} \\ K_{ca} & K_{cc} \end{bmatrix} \begin{bmatrix} f_a \\ f_c \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix} \tag{2.8}$$

where $f_a = K_a^{-1}$ and then use

$$\begin{bmatrix} \varphi_a \\ \varphi_c \end{bmatrix} = \begin{bmatrix} I \\ f_c \; K_a \end{bmatrix} \varphi_a \tag{2.9}$$

Although the degrees of freedom have been partitioned in Eqs. (2.8) and (2.9), there is obviously no need for it in this analysis. Note that instead of Eq. (2.7) we would now consider the problem $\frac{1}{\omega^2}\varphi_a = f_a\, m_a\, \varphi_a$ (see Eq. (2.17)).

The order of Eq. (2.7) is the number of mass degrees of freedom allocated to the structure. Depending on the engineer's experience and the structure analyzed, the eigenvalues obtained from this equation may only be crude approximations to the eigenvalues of the original model of Eq. (2.3).

A more general technique for finding an approximation to the lowest eigenvalues and vectors of large systems is the Rayleigh Ritz analysis. For a general discussion of this method we consider the eigenvalue problem

$$Av = \lambda Bv \tag{2.10}$$

with A and B positive definite and the operators defined in an n-dimensional space V_n. The Rayleigh minimum principle states that

$$\lambda_1 = \min \rho(v) \tag{2.11}$$

where the minimum is taken over all functions v and $\rho(v)$ is the Rayleigh quotient

$$\rho(v) = \frac{(v,Av)}{(v,Bv)} > 0 \tag{2.12}$$

In the Ritz analysis we define a set of funtions $\bar{v}$ in a subspace V_q of dimension q

$$\bar{v} = \sum_{i=1}^{q} \xi_i f_i \tag{2.13}$$

where the f_i are the Ritz basis functions and the ξ_i are Ritz coordinates. Substituting Eq. (2.13) into (2.12) we get

$$\rho(\bar{v}) = \frac{\sum_{j=1}^{q} \sum_{i=1}^{q} \xi_i \xi_j \tilde{a}_{ij}}{\sum_{j=1}^{q} \sum_{i=1}^{q} \xi_i \xi_j \tilde{b}_{ij}} \tag{2.14}$$

with

$$\begin{aligned} \tilde{a}_{ij} &= (f_i, Af_j) \\ \tilde{b}_{ij} &= (f_i, Bf_j) \end{aligned} \tag{2.15}$$

The necessary condition for a minimum of $\rho(\bar{v})$ is $\frac{\partial \rho}{\partial \xi_i} = 0$ (i=1,...,q). This yields

$$\tilde{A}a = \rho \tilde{B}a \tag{2.16}$$

where a is a vector listing the Ritz coordinates, $\tilde{A}$ and $\tilde{B}$ are full symmetric matrices with typical elements given in Eq. (2.15).

The solution of Eq. (2.16) yields q eigenvalues $\rho_1,\ldots,\rho_q$ and corresponding eigenvectors, which are used to obtain from Eq. (2.13) $\bar{v}_1,\ldots,\bar{v}_q$. The eigenvalues are upper bound approximations to the eigenvalues of Eq. (2.10), i.e.

$$\lambda_1 \le \rho_1 \; ; \; \lambda_2 \le \rho_2 \; ; \; \lambda_3 \le \rho_3 \; ; \; \ldots \; ; \; \lambda_q \le \rho_q$$

The first inequality is obvious because V_q is contained in V_n. To prove the second inequality we observe that

$$\lambda_2 = \min \frac{(v,Av)}{(v,Bv)}$$

with the constraint

$$(v,Bv_1) = 0$$

Similarly

$$\rho_2 = \min \frac{(\bar{v},A\bar{v})}{(\bar{v},B\bar{v})}$$

satisfying

$$(\bar{v},B\bar{v}_1) = 0$$

Now consider an auxiliary problem, in which

$$\tilde{\rho}_2 = \min \frac{(\bar{v},A\bar{v})}{(\bar{v},B\bar{v})}$$

with the constraint

$$(\bar{v},Bv_1) = 0$$

We realize that $\lambda_2 \le \tilde{\rho}_2$ because V_q is contained in V_n. But $\tilde{\rho}_2 \le \rho_2$ since the most severe constraint on $\bar{v}$ is the eigenfunction $\bar{v}_1$. Therefore

$$\lambda_2 \le \tilde{\rho}_2 \le \rho_2$$

The inequalities for λ_3 to λ_q are proved similarly.

In dynamic analysis we obtain the Ritz functions from a static solution in which q load patterns are specified in R, i.e.

$$KT = R$$

and for Eq. (2.16)

$$\tilde{A} = T^T R \; ; \; \tilde{B} = T^T M T$$

Although we have shown that an eigenvalue calculated from a Ritz analysis is an upper bound on the corresponding exact eigenvalue of the system, we did not establish anything about the error in the eigenvalue. Naturally this error depends on the Ritz functions chosen. We only obtain good results if the functions span a subspace which is close to the least dominant invariant q-dimensional subspace of the operators.

The technique of mass lumping followed by static condensation and the Ritz analysis have been presented as two methods. Together with other techniques, such as the component mode synthesis [3], they are recognized in Chapter 4 as the first step in a subspace iteration. Let us show that the static condensation procedure resulting in Eq. (2.7) is actually a Ritz analysis. It is demonstrated in Section 3.6 that zero diagonal elements in the mass matrix correspond to infinite frequencies. To obtain approximations to the lowest frequencies in Eq. (2.6) we can apply the Ritz analysis. The Ritz functions are the displacement patterns associated with the degrees of freedom φ_a, and are given in Eq. (2.9). Transforming Eq. (2.6) we obtain

$$\begin{bmatrix} I \\ f_c K_a \end{bmatrix}^T \begin{bmatrix} K_{aa} & K_{ac} \\ K_{ca} & K_{cc} \end{bmatrix} \begin{bmatrix} I \\ f_c K_a \end{bmatrix} \varphi_a = \omega^2 \begin{bmatrix} I \\ f_c K_a \end{bmatrix}^T \begin{bmatrix} m_a & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} I \\ f_c K_a \end{bmatrix} \varphi_a$$

or

$$K_a \varphi_a = \omega^2 m_a \varphi_a$$

which is Eq. (2.7). Therefore, in the static condensation we perform a Ritz analysis of the lumped mass model of the structure. Because the Ritz functions span the subspace which corresponds to the finite eigenvalues of the model, we calculate these eigenvalues 'exactly' (see Section 4.2). If less Ritz functions are used, upper bounds to the lowest eigenvalues are obtained as discussed above. In practice, the equivalent Ritz analysis would be carried out using the transformation

$$\begin{bmatrix} \varphi_a \\ \varphi_c \end{bmatrix} = T\, a \; ; \; T = \begin{bmatrix} f_a \\ f_c \end{bmatrix}$$

where f_a and f_c are calculated in Eq. (2.8), and the vector a lists the Ritz coordinates. We note that $a = K_a \varphi_a$. The Ritz transformation of Eq. (2.6) gives

$$f_a\, a = \omega^2 f_a\, m_a\, f_a\, a \qquad (2.17)$$

In the Ritz analysis, hardly more numerical effort is required if the original model of Eq. (2.3) is analyzed and if in Eq. (2.8) different load patterns are specified in order to obtain a 'better' T. If mass is lumped in the analysis of a complex structure, the number of mass degrees of freedom should still be significantly larger than the number of eigenvalues required, in order to keep an adequate mass distribution in the system.

The main difficulty in a Ritz analysis of a complex structure is the selection of 'good' basis functions. It is often thought

that a repetition of the analysis with a somewhat larger set of Ritz functions is a good check on the results of the first analysis. This is not necessarily true. In repeating the analysis we may merely detect that our first analysis gave bad approximations to the lowest eigenvalues and vectors.

In summary, two serious problems are present. Firstly, we cannot estimate how accurate our approximations to the required eigenvalues and vectors are. Secondly, we do not know if we miss an approximation to a lower eigenvalue and vector altogether. This uncertainty in a practical dynamic analysis may lead to a large number of repetitions of the analysis, involving a high cost which nevertheless does not remove all uncertainty. It may then have been more efficient to rather solve once and for all accurately for the required eigenvalues and vectors.

The structural model of Eq. (2.3) may be a refined representation of the structure or a rather crude stiffness and mass approximation. In my opinion, programs should be available to analyze any model, without the analyst fitting it to particular program requirements.

In earthquake analysis we mainly require the lowest modes of the system. In other important problems in dynamics we may need to find all eigenvalues and associated vectors in a given interval. This may be the case, because of a high level of power spectral density of excitation in a given frequency region [4].

The large eigenvalue problems arising in dynamic analysis and considered here, can therefore be stated as follows.

Given two operators A and B, find

1. the lowest few λ and associated v which satisfy

$$Av = \lambda Bv \tag{2.18}$$

or

2. find all λ and corresponding v which satisfy Eq. (2.18) and where $\mu_B < \lambda < \mu_T$.

In Eq. (2.18) the operator A is of order n, symmetric and positive definite and has bandwidth $(2m_A + 1)$. The operator B has either the same properties as A or is diagonal and non-negative definite. The eigenvalues in Eq. (2.18) are all positive. Also, we consider the problem as large if it is much cheaper to solve for only the required eigenvalues and vectors instead of calculating simply all. In general, the system will be large if the high speed core storage of a reasonable size computer is too small to use an in-core Householder-QR-Inverse iteration technique [1] (see Appendix I and Section 6.7)

2.3 The Eigenvalue Problem in Buckling Analysis

The equations governing bifurcation buckling of a structure are

$$Ku = \lambda K_G u \tag{2.19}$$

where K is the small deflection stiffness matrix and K_G is the geometric stiffness matrix of the system [4]. The parameter λ gives the buckling load and u is the corresponding buckling mode.

We observe that this is an eigenvalue problem of the form stated in the previous section. However, in this case the operator B, which corresponds to K_G in Eq. (2.19), is in general indefinite and is always banded. Using the notation in Eq. (2.18) we consider the problem

$$Bv = \varkappa Av \tag{2.20}$$

in which $\varkappa = 1/\lambda$ and can be negative or positive. In this equation we want to solve for the maximum value of $\varkappa$ which gives the lowest buckling load. It may also be of interest to find the next lowest buckling loads. Namely, if they are very close, then preventing the lowest buckling mode to occur does not make the structure much safer.

For solution of the buckling problem we naturally cannot use the static condensation analysis. A Ritz analysis is appropriate, but the problems discussed in the previous section are again present.

3. A DETERMINANT SEARCH TECHNIQUE FOR THE SOLUTION OF $Av = \lambda Bv$ WITH SMALL BANDWIDTH

3.1 Introduction

Determinant search techniques have been used since long ago for finding the eigenvalues of small symmetric and nonsymmetric matrices [5]. For the analysis of larger problems in structural engineering the technique was generally considered inefficient, because each determinant evaluation requires one triangular factorization. If a determinant search iteration is used alone to calculate eigenvalues to high precision, many factorizations may be necessary. This is costly unless the bandwidth of the system is very small.

In this chapter a very efficient algorithm which combines triangular factorization with vector iteration is presented. A determinant search is only used to shift into the vicinity of the next unknown root. As will be shown, the number of negative pivots in a factorization tells if an unknown root is smaller than the current shift. Inverse iteration is then used to find the vector and eigenvalue.

Programs SECANT and SECANTD use the technique and are briefly introduced in Section 3.7. Example analyses are given in Chapter 6.

3.2 Considerations for an Iteration Scheme

We first consider the generalized eigenvalue problem

$$Av = \lambda Bv \tag{3.1}$$

when B is positive definite. The case B diagonal non-negative definite is discussed in Section 3.6. An equivalent problem is

to calculate the zeros of the polynomial $p(\lambda) = \det (A - \lambda B)$. Our aim is to find a few lowest eigenvalues and associated vectors when A and B are large and have small bandwidth.

For solution we may think of some well-known techniques. Let us briefly mention them.

A transformation of Eq. (3.1) to the standard eigenvalue problem yields

$$\bar{A}\,\bar{v} = \lambda\,\bar{v} \tag{3.2}$$

where

$$\bar{A} = \tilde{L}^{-1}\,A\,\tilde{L}^{-T} \tag{3.3}$$

$$B = \tilde{L}\,\tilde{L}^{T} \tag{3.4}$$

$$\bar{v} = \tilde{L}^{T}\,v \tag{3.5}$$

In Eq. (3.4) we find the Cholesky decomposition of B. Therefore B should not be ill-conditioned with respect to inversion (see Section 4.3).

Let us distinguish two cases: Assume that B is banded. Then $\bar{A}$ is full. Therefore, regarding storage and the solution of the eigenvalue problem in Eq. (3.1) the transformation is uneconomical.

However, if B is diagonal, we find the transformation trivial and $\bar{A}$ has the same band as A. We now need to solve a standard eigenvalue problem.

As $\bar{A}$ is large, a Householder-QR-Inverse iteration solution is uneconomical because we would set out to find all eigenvalues without taking advantage of the band in $\bar{A}$.

An efficient technique to find an eigenvalue and the associated eigenvector of $\bar{A}$ is the Rayleigh quotient iteration defined as

$$(\bar{A} - \rho(x_k)\ I)\ x_{k+1} = x_k\ \ell_k \qquad k = 1,2,... \tag{3.6}$$

$$\rho(x_k) = \frac{(x_k,\ \bar{A}x_k)}{(x_k,\ x_k)} \tag{3.7}$$

where ℓ_k is chosen to normalize x_{k+1}. Under certain conditions convergence is ultimately cubic to an eigenpair $(\lambda_i, \bar{v}_i)$ [6] [22]. The particular eigenvalue to which the iteration converges depends on the initial vector chosen. We note that each iteration needs one triangular factorization, and that we may well converge to an eigenvalue which we are not interested in. For our problem, it is much more efficient to apply a shift only after always a few inverse iterations and to assure that the eigenvalues are found sequentially from the lowest one upwards.

This technique is used in program BANEIG [7]. The algorithm finds the smallest eigenvalue and corresponding vector of $\bar{A}$ by shifting from the left towards λ_1. This way the shifted matrix remains positive definite. The shift is determined using an empirical rule once the iteration vector has settled down in the inverse iteration. After calculation of λ_1 and $\bar{v}_1$, the algorithm uses an orthogonal similarity transformation due to Rutishauser to deflate the matrix [8]. The importance of this deflation is that the new matrix has still the same band as $\bar{A}$. However, the eigenvector needs to be found to high precision. Inverse iteration with shifting as before yields the next

smallest eigenvalue. The program continues with deflation and inverse iteration until all required eigenvalues have been calculated. The final eigenvectors of Eq. (3.2) are obtained by applying the orthogonal transformations back onto the eigenvectors of the deflated matrices.

As stated before the program can handle only the case B diagonal and positive definite. Numerical difficulties can arise if some diagonal elements in B are small.

Basically, the algorithm uses triangular factorization and inverse iteration. The undesirable matrix deflation is necessary to assure convergence to an eigenvalue and vector not yet calculated. The transformation to the standard eigenvalue problem must be performed to be able to use the deflation procedure.

A more direct scheme would use triangular factorization and inverse iteration on Eq. (3.1) without transforming to the standard eigenvalue problem. Then the case B banded would not present extra numerical problems. But to obtain best program efficiency we should use efficiently all the information that we can obtain from each factorization and inverse iteration.

3.3 The Triangular Factorization and its Use in the Iteration Scheme

In inverse iteration with Rayleigh quotient shifts, triangular factorization is basically used to speed convergence in the vector iteration. Essentially the iteration goes for the vector but the eigenvalue is found at the same time.

To benefit more from a factorization we can evaluate the characteristic polynomial at the current shift μ_k.

Let $p(\mu) = \det(A - \mu B)$ and assume that we have the LDL^T decomposition of $(A - \mu_k B)$, then

$$LDL^T = (A - \mu_k B) \tag{3.8}$$

$$p(\mu_k) = \det(LDL^T) = \prod_{i=1}^{n} d_{ii} \tag{3.9}$$

The polynomial values at successive shifts can directly be used to iterate towards a root. However, a clever scheme must be implemented to assure convergence to the next unknown root.

Another most important observation can be used at each factorization. Simply stated, we know that the number of negative elements in D in Eq. (3.8) is equal to the number of eigenvalues smaller than μ_k.

This statement and the polynomial iteration scheme are explained in detail after the triangular factorization has been discussed.

3.3.1 The Triangular Factorization of $A - \mu_k B$

We note that for μ_k larger than λ_1, the matrix $(A - \mu_k B)$ is not positive definite. Therefore, we cannot find its Cholesky factors but the LDL^T decomposition exists provided none of the leading principal minors vanishes.

A practical way of obtaining L and D in Eq. (3.8) is to use simple Gaussian elimination on $A - \mu_k B$. Here we reduce the matrix into upper triangular form expressed as

$$L_{n-1}^{-1} \cdots L_2^{-1} \; L_1^{-1} (A - \mu_k B) = U \tag{3.10}$$

where

$$L_k^{-1} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & -\ell_{k+1,k} & \\ & & -\ell_{k+2,k} & \ddots \\ & & \vdots & \\ & & -\ell_{k+m_A,k} & \ddots \\ & & & 1 \end{bmatrix}; \quad \ell_{k+i,k} = \frac{c^{(k)}_{k+i,k}}{c^{(k)}_{kk}}; \quad u_{kk} = c^{(k)}_{kk}$$

and $c^{(k)}_{\ell m}$ denotes the (ℓ,m) element of the matrix $C = A - \mu_k B$ after the first (k-1) row reductions have been carried out. Equation (3.10) rewritten becomes Eq. (3.8) where

$$L = L_1 L_2 \cdots L_{n-1} \; ; \; DL^T = U$$

We note that the k'th leading principal minor M_k is given as

$$M_k = u_{11} u_{22} \cdots u_{kk} \tag{3.11}$$

Obviously, if M_{k-1} is nonzero, but M_k is zero, then u_{kk} equals zero, and the decomposition does not exist, also indicated by the fact that multipliers $\ell_{k+i,k}$ become infinitely large. In practice, a decomposition is regarded as numerically unstable, if multiplier growth occurs. In such case the errors involved in the decomposition are large. In general, partial pivoting could be used to ensure that multipliers do not exceed unity in modulus. However, the bandwidth of the system would then increase and more operations are required [5].

In the programs the triangular factorization in Eq. (3.10) is used. As described above, the main aim in the eigenvalue iteration is to shift into the vicinity of the next unknown root.

If the triangular factorization would prove to be unstable, the program would have to increase the shift in its last digits and try a new factorization. In all example analyses this never happened. Also, experiments have been carried out, in which matrices have been triangularized at and near calculated eigenvalues. The decompositions have always proved stable. The discussion in the next section indicates why instability is unlikely to occur.

As an estimate of the work involved in the evaluation of Eq. (3.8) plus the calculation of the determinant, we consider the number of operations required. One operation is equal to one multiplication which is nearly always followed by an addition. Assume that the half bandwidths m_A and m_B are full and constant. Neglecting terms involving the bandwidths only, we have as the number of operations required

$$\frac{1}{2} n m^2 + \frac{5}{2} n m + 2n \quad \text{when} \quad m = m_A = m_B$$

$$\frac{1}{2} n m^2 + \frac{3}{2} n m + 2n \quad \text{when} \quad m = m_A;\ m_B = 0$$

These formulae are used for comparison purposes. In most actual systems the bandwidths vary and many zero multipliers occur. The solution solver must take due advantage of both.

3.3.2 The Eigenvalue Separation Theorem

Consider the problem $Av = \lambda Bv$ in Eq. (3.1) and a different problem of dimension n - 1 which we call the first associated constraint problem, i.e.

$$A'v' = \lambda' B'v' \tag{3.12}$$

where A' and B' are obtained by omitting the last rows and columns of A and B. We can show that the eigenvalues of this problem are separating those of Eq. (3.1), i.e.

$$\lambda_r \leq \lambda'_r \leq \lambda_{r+1} \tag{3.13}$$

In a dynamic analysis this means that the frequencies of a structure which is constrained in its n'th degree of freedom lie in between the frequencies of the unconstrained structure. For a simple and elegant proof of Eq. (3.13) we use the minimax characterization of eigenvalues [9], which says

$$\lambda_{r+1} = \max\left\{\min \frac{(v,Av)}{(v,Bv)}\right\} \tag{3.14}$$

with v satisfying $(g_i,v) = 0$ $(i = 1,\ldots,r)$, where the g_i are arbitrary constraint vectors. Equation (3.14) simply states that we select a set of g_i and find the minimum of the Rayleigh quotient satisfying the constraints over all v. The maximum of these minima as the constraint vectors are varied equals λ_{r+1}.

Similarly, for the problem in Eq. (3.12)

$$\lambda'_r = \max\left\{\min \frac{(v,Av)}{(v,Bv)}\right\}$$

with $(g_i,v) = 0$ $(i=1,\ldots,r)$ where the g_i are arbitrary for $i=1,\ldots,r-1$ but $g_r = e_n$. This ensures that the last element in v is zero, because e_n is the last column in the n×n identity matrix I. Because the constraint for λ_{r+1} can be more severe and includes that for λ'_r, we have

$$\lambda'_r \leq \lambda_{r+1}$$

To determine λ_r we use

$$\lambda_r = \max\left\{\min \frac{(v,Av)}{(v,Bv)}\right\}$$

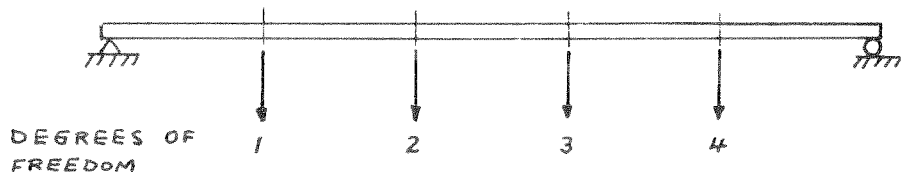
with $(g_i,v) = 0$ $(i=1,\ldots,r-1)$ and all g_i are arbitrary. To evaluate λ_r' we have the same constraints but one more, hence

$$\lambda_r \leq \lambda_r'$$

which proves Eq. (3.13). In the same way, the eigenvalues of the second constraint problem obtained by eliminating the last two rows and columns of A and B separate those of the first constraint problem, etc.

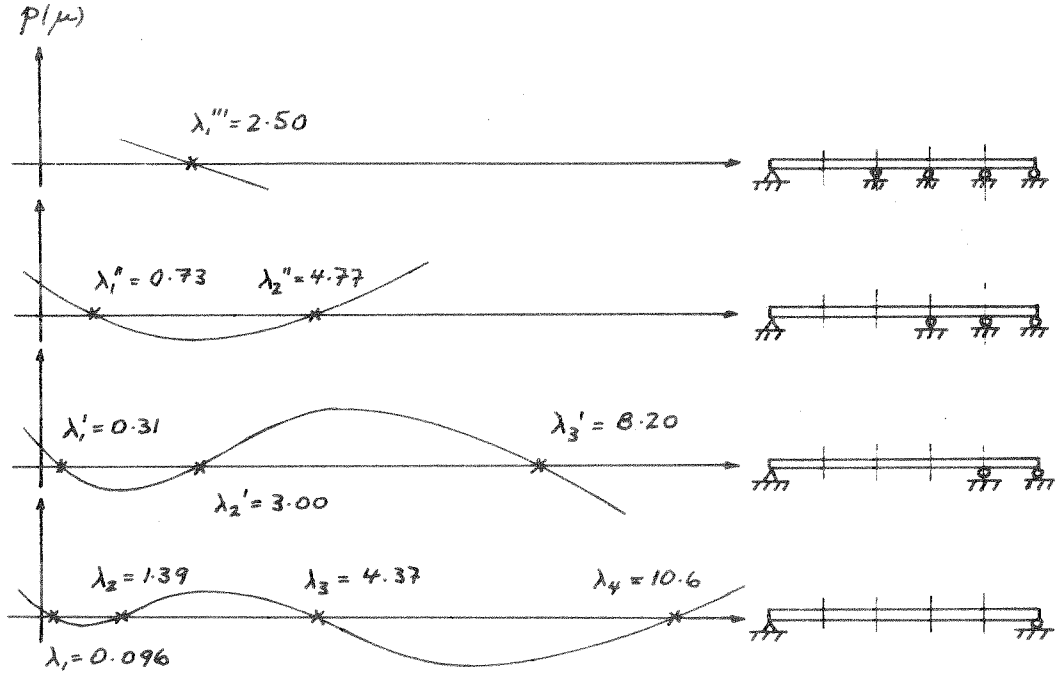
Figure 1 gives the eigenvalues of a simply supported beam with four degrees of freedom and of its constraint problems. Recall that a sequence of polynomials $\left\{\varphi_i(x)\right\}_{i=1,\ldots,n}$ form a Sturm Sequence if the roots of the polynomial $\varphi_i(x)$ separate the roots of $\varphi_{i+1}(x)$. Hence we have proved that the characteristic polynomials of the associated constraint problems and of the eigenvalue problem in Eq. (3.1) form a Sturm Sequence.

We can now give a reason why instabilility in the triangular factorization of Eq. (3.10) is unlikely to occur. A small i'th pivot in the factorization means that the current shift is an eigenvalue of the constraint problem obtained by fixing the degrees of freedom (i+1) to n. But from experience we know that the frequencies of vibration of the structure obtained by fixing many degrees of freedom are much higher than the lowest frequencies of the unconstrained structure. Hence, if at all, a small pivot is likely to occur only when i is near to n. This means that multiplier growth is not possible.



CHOOSE BEAM PROPERTIES TO OBTAIN USING FINITE DIFFERENCES :

$$A = \begin{bmatrix} 5 & -4 & 1 & \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ & 1 & -4 & 5 \end{bmatrix} \quad ; \quad B = \begin{bmatrix} 2 & & & \\ & 2 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$



EIGENVALUES AND SKETCH OF CHARACTERISTIC POLYNOMIALS

PROBLEM

FIGURE 1: EIGENVALUES OF SIMPLY SUPPORTED BEAM AND OF ASSOCIATED CONSTRAINT PROBLEMS

The practical use of the eigenvalue separation property is as follows: Assume that we carry out a triangular factorization at shift μ_k and that μ_k is not an eigenvalue of the (n-1) associated constraint problems. Then we can use D in Eq. (3.8) to calculate the determinants of all constraint problems. But consider only their signs. Referring to Fig. 1, we find that due to the eigenvalue separation property, for $\lambda_r < \mu_k < \lambda_{r+1}$ we must have exactly r negative elements in D. Hence at any shift μ_k, the number of negative elements in D tells us how many eigenvalues in Eq. (3.1) are smaller than μ_k. This is a most important fact, because with it we are able to find how many eigenvalues exist in a particular interval.

The Sturm Sequence property has been used in simple bisection to find eigenvalues of matrices [10][11]. However, this is only economical if the matrices have very small bandwidth, in particular when A and B are tridiagonal. A more efficient algorithm would use the polynomial values at the shifts to accelerate iteration to the next unknown root. The eigenvalue separation property can be used at each shift to check if an unknown root has been passed.

3.3.3 The Interpolation Scheme

Consider the iteration to λ_1 in Fig. 2 where iterates μ_k and μ_{k-1} are lower bounds to the root. As will be shown in Section 3.5 it is most economical for us to obtain merely a shift near λ_1 and then start inverse iteration for the eigenvector. The extrapolation formula used in the algorithm is

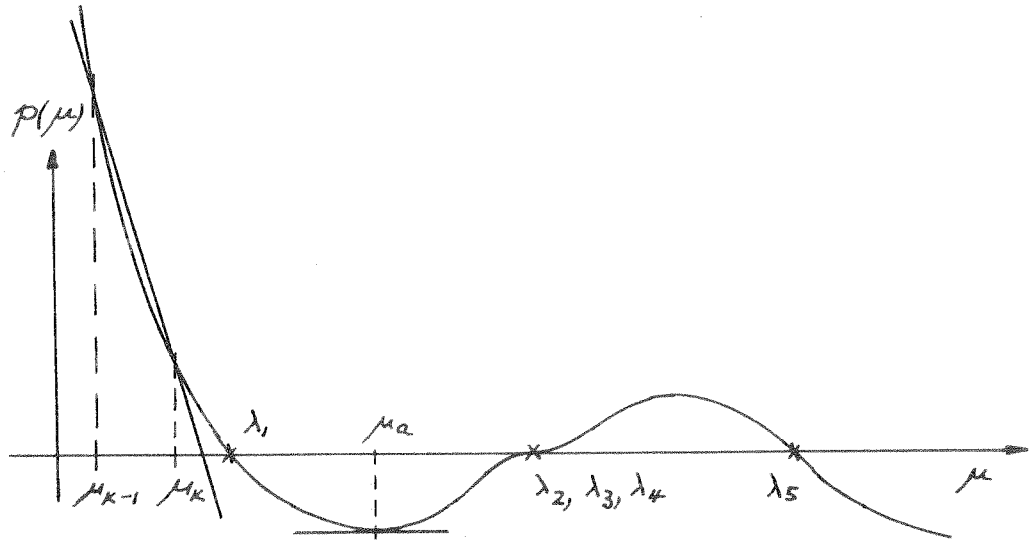


FIGURE 2: CHARACTERISTIC POLYNOMIAL $p(\mu)$

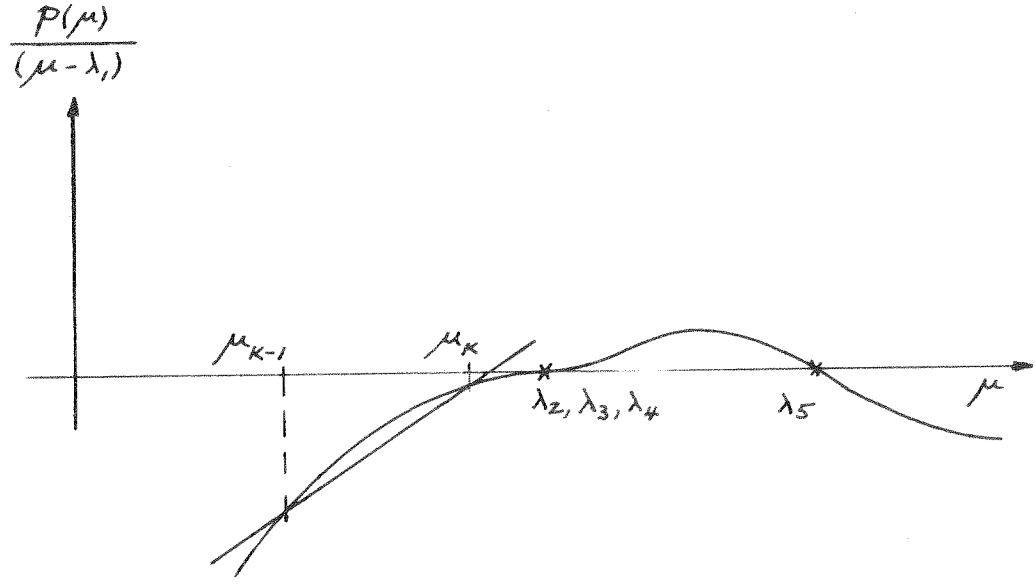


FIGURE 3: $p(\mu)$ WITH λ_1 SUPPRESSED

$$\mu_{k+1} = \mu_k + \eta \frac{p(\mu_k)}{p(\mu_k) - p(\mu_{k-1})} (\mu_{k-1} - \mu_k) \qquad (3.15)$$

with η a constant. When $\eta = 1.0$ we have the well-known Secant iteration, where $\mu_{k+1} \leq \lambda_1$ and $\mu_{k+1} \to \lambda_1$ as $k \to \infty$. Convergence in this iteration can be slow, and we need an efficient acceleration scheme. Starting the iteration, we let $\eta = 2.0$ because it is known that in this case $\mu_{k+1} \leq \mu_a$, where μ_a is the smallest stationary point of p [5]. A jump over a root would simply be detected by a sign change in p. However, when we iterate towards a multiple root as in Fig. 3, convergence with $\eta = 2.0$ is still slow. Fortunately, the eigenvalue separation theorem allows us to accelerate the iteration still more. We double η after each iteration in which the iterates did not change in their first 3 to 4 digits. We may thus jump over a single root, a multiple root or into a cluster of roots, but this is always detected by counting the number of negative elements in D. Naturally, with the strategy adopted, we cannot jump far beyond the unknown roots.

The advantage of the one-sided approach to λ_1 is also obtained for any other root, say λ_{j+1}, by using instead of $p(\mu_k)$ in Eq. (3.15), the deflated polynomial $p_j(\mu_k)$, where

$$p_j(\mu_k) = p(\mu_k) / \prod_{i=1}^{j} (\mu_k - \lambda_i) \qquad (3.16)$$

and the λ_1 to λ_j have already been calculated, Fig. 3.

In the algorithm the accelerated Secant iteration is stopped once either of the following criteria is satisfied:

(1) Assume that we did not jump over a root and that the correction to μ_k in Eq. (3.15) is smaller than one half of the final tolerance required on the root. We then approached the root from below and are near enough to start inverse iteration. The final eigenvalue is calculated by adding the Rayleigh correction to μ_k (see Section 3.4).

(2) Let α be the number of negative pivots in the factorization at μ_{k+1}, and β be the number of eigenvalues which have been determined and which are smaller than μ_{k+1}. Suppose $\gamma = \alpha - \beta$, then we know that we jumped over γ unknown eigenvalues. Inverse iteration is now used for the vectors and Rayleigh corrections are calculated to obtain the γ eigenvalues. In the process eigenvectors corresponding to eigenvalues larger than μ_{k+1} may be found; for example, if we jumped into an eigenvalue cluster.

For iteration towards the next unknown eigenvalue, we use Eq. (3.16) to suppress the last found roots from two previously calculated polynomial values, which are the starting points in Eq. (3.15) with $\eta = 2.0$. In Eq. (3.16) we do not want to divide by values close to zero and therefore select two μ-values far enough from the calculated roots. As implemented in the programs, it is only necessary to store the last three calculated polynomial values.

The objective of obtaining economically a shift near the next unknown root can also be pursued using a Newton iteration with the same factor η, where

$$\mu_{k+1} = \mu_k - \eta \frac{p(\mu_k)}{p'(\mu_k)} \tag{3.17}$$

The Newton iteration has been used efficiently in the eigenvalue solution of tridiagonal matrices [5]. For matrices with larger bandwidth the main difficulty lies in finding an economical algorithm to evaluate $p'(\mu_k)$. The following scheme was considered. The function $p'(\mu)$ is given by

$$p'(\mu) = \det B \frac{d}{d\mu} \det (B^{-1}A - \mu I) \tag{3.18}$$

where μ is now a variable.

FACT Let K be a square matrix of order n, with its elements k_{ij} functions of t. Then

$$\frac{d}{dt} \det K = \sum_{j=1}^{n} \det K^{(j)}$$

where $K^{(j)}$ is K except in row j, which is now $(k'_{j1} \quad k'_{j2} \ldots k'_{jn})$. Using the fact we have

$$p'(\mu) = \det B\left\{- \sum_{j=1}^{n} \det(B^{-1}A - \mu I)^{(j,j)}\right\} \tag{3.19}$$

where $(B^{-1}A - \mu I)^{(j,j)}$ is obtained by deleting row j and column j from $(B^{-1}A - \mu I)$.

We need to use an efficient scheme to evaluate Eq. (3.19). Consider an equation for element (j,j) in $(B^{-1}A - \mu I)^{-1}$,

$$\{(B^{-1}A - \mu I)^{-1}\}_{jj} = \frac{\det(B^{-1}A - \mu I)^{(j,j)}}{\det(B^{-1}A - \mu I)}$$

Hence Eq. (3.19) becomes

$$p'(\mu) = - \det B\left\{\det(B^{-1}A - \mu I) \sum_{j=1}^{n} \left\{(B^{-1}A - \mu I)^{-1}\right\}_{jj}\right\}$$

and for $\mu = \mu_k$

$$p'(\mu_k) = - \det(A - \mu_k B) \sum_{j=1}^{n} \left\{(B^{-1}A - \mu_k I)^{-1}\right\}_{jj} \qquad (3.20)$$

To calculate $\left\{(B^{-1}A - \mu_k I)^{-1}\right\}_{jj}$ we solve

$$(A - \mu_k B)\, z = B e_j \quad (j=1,\ldots,n) \qquad (3.21)$$

where we need to find only the j'th element in z, say z_j. The Newton iteration formula, Eq. (3.17), then becomes

$$\mu_{k+1} = \mu_k + \frac{\eta}{\sum_{j=1}^{n} z_j}$$

One may like to compare the cost of two Secant steps with one Newton iteration. Essentially, we then need to compare the operations involved in the solution of Eq. (3.21) and two factorizations or about $n^2\, m_A$ versus $\frac{1}{2}\, n\, m_A^2$ operations. Therefore the Newton iteration is much more costly. The algorithm was tested and it was found that the iteration times are much longer than using Secant steps.

3.3.4 Starting Iteration Values

In the Secant iteration we need two starting values μ_1 and μ_2, which are both lower bounds on λ_1. With B positive definite one starting value is zero. As the second value we may use a

negative number, whose magnitude depends on the norms of A and B. However, it is more efficient to use for μ_2 a positive lower bound on λ_1. If μ_2 is close to λ_1 we may need only one more factorization to bracket the root. Let us note that a positive lower bound on λ_1 cannot be obtained using matrix norms only. Assume that A is a structure stiffness matrix with the rigid body modes not removed, and that B = I. It is clear that from the magnitudes of the elements in A alone we cannot conclude that the lowest eigenvalue of A is zero.

A positive lower bound estimate on λ_1 would be obtained from a single Newton step. But the evaluation of the tangent is costly. Therefore, the following scheme was used. An approximate tangent was evaluated using first 3, then 5, then 7, etc. evenly spaced columns in B, until successive tangent approximations varied by less than a factor. If the factor is small, we probably have to evaluate the tangent quite accurately. But if the factor is large, then μ_2 may be much larger than λ_1. In this case, if there are γ negative pivots in the factorization at μ_2 the next estimate for μ_2 is taken equal to the last one divided by $(\gamma + 1)$ until $\gamma = 0$. It is reasonable to use in the approximate tangent evaluation about as many operations as in a single Secant step. On this basis we should use approximately $\frac{1}{2}$ m_A columns in B. However, as also observed in test examples, in general many more columns in B may be needed to obtain from the approximate Newton step $\mu_2 < \lambda_1$.

not more effective.

Inverse iteration analysis shows that an iteration vector ultimately converges to the eigenvector corresponding to the eigenvalue nearest to the shift, provided the starting vector is not deficient in that eigenvector. But it is important to note that with a low convergence tolerance, we may accept an approximation to any other eigenvector. In fact, if the tolerance is too low we may accept a vector which is not an approximation to an eigenvector at all. Also, convergence should be based on the Rayleigh corrections calculated. If the vector iterates into an eigenspace of dimension larger than one, two successive iteration vectors may be completely different and yet have converged.

For an operation count we summarize an inverse iteration step at the shift μ to consist of

$$(A - \mu B)\ \bar{x}_{k+1} = y_k \tag{3.24}$$

where referring to Section 3.3.1 we obtain $\bar{x}_{k+1}$ using

$$LDL^T\ \bar{x}_{k+1} = y_k$$

The right hand side reduction gives

$$L^T\ \bar{x}_{k+1} = D^{-1}\ L^{-1}\ y_k$$

and the back-substitution gives $\bar{x}_{k+1}$ (see also Chapter 5). Next we form

$$\bar{y}_{k+1} = B\ \bar{x}_{k+1} \tag{3.25}$$

$$\rho^c(\bar{x}_{k+1}) = \frac{(\bar{x}_{k+1}, y_k)}{(\bar{x}_{k+1}, \bar{y}_{k+1})} \tag{3.26}$$

$$y_{k+1} = (\bar{y}_{k+1} - \alpha_1 w_1 - \ldots \alpha_t w_t)/(\bar{x}_{k+1}, \bar{y}_{k+1})^{\frac{1}{2}} \qquad (3.27)$$

where the w_i are stored in core, and

$$w_i = Bv_i$$

$$\alpha_j = (\bar{y}_{k+1}, v_j)$$

The number of operations are

$$4n\,m + 2nt + 5n \quad \text{when} \quad m = m_A = m_B$$

$$2n\,m + 2nt + 5n \quad \text{when} \quad m = m_A \; ; \; m_B = 0$$

If the eigenvectors are calculated to high precision we would only need to orthogonalize the starting iteration vector to the eigenvectors already found. However, in general, the orthogonalization in each iteration is advisable.

We should mention an inverse iteration scheme in which we use the transformation $z_k = \tilde{L}^T x_k$ with $B = \tilde{L}\tilde{L}^T$. Then Eq. (3.24) can be written as

$$\{\tilde{L}^{-1}(A - \mu B)\,\tilde{L}^{-T}\}\,\bar{z}_{k+1} = z_k \qquad (3.28)$$

and we iterate as follows

$$\tilde{z}_k = \tilde{L}\, z_k$$

$$(A - \mu B)\tilde{\tilde{z}}_{k+1} = \tilde{z}_k$$

$$\bar{z}_{k+1} = \tilde{L}^T \tilde{\tilde{z}}_{k+1}$$

$$\rho^c(\bar{z}_{k+1}) = \frac{(\bar{z}_{k+1}, z_k)}{(\bar{z}_{k+1}, \bar{z}_{k+1})}$$

$$z_{k+1} = (\bar{z}_{k+1} - \beta_1 \bar{v}_1 - \ldots - \beta_t \bar{v}_t)/(\bar{z}_{k+1}, \bar{z}_{k+1})^{\frac{1}{2}}$$

where

$$\beta_j = (\bar{z}_{k+1}, \bar{v}_j)$$

$$\bar{v}_j = \tilde{L}^T v_j$$

There is practically no difference in the number of operations required, but we only need storage for the vectors $\bar{v}_i$, whereas using Eq. (3.24) to (3.27) we store the v_i and w_i. However, we require the Cholesky factor of B. The direct scheme in Eq. (3.24) to (3.27) is preferable because there are no difficulties when B is ill-conditioned or non-negative definite.

3.5 Computational Aspects

In the preceding sections the algorithm was described to perform Secant steps in order to shift into the vicinity of the next unknown root. Then the iteration for the vector is started. The final eigenvalue is obtained by adding the Rayleigh correction to the shift. Alternatively, one may calculate the root in a Secant iteration to high precision and obtain the vector in one or two inverse iterations. The greater efficiency of one method over the other depends largely on the work involved in a Secant step and in an inverse iteration.

Figure 4 shows the number of inverse iterations equivalent in operations to one Secant step. In practical structural analysis the half bandwidth is seldom smaller than 20 or 30, and then inverse iteration is already considerably cheaper than triangular factorization. After the jump, we may need, if many, 8 inverse iterations to find the final Rayleigh correction and the eigenvector. At a small, practical bandwidth, the operations

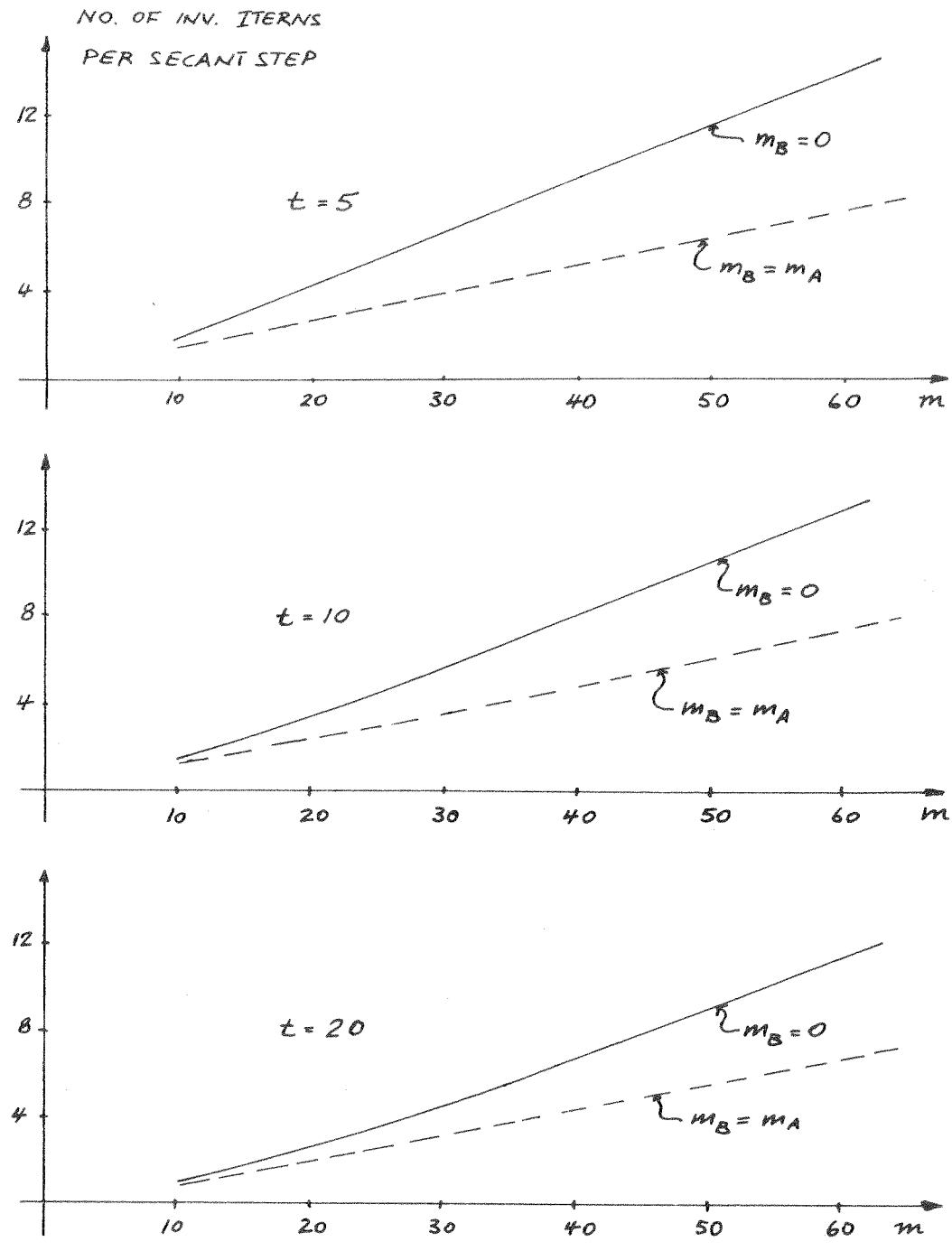


FIGURE 4 : NUMBER OF INVERSE ITERATIONS PER SECANT STEP

t = NUMBER OF VECTORS ALREADY CALCULATED

required would be equivalent to about two factorizations. Therefore, the strategy for starting inverse iteration is economical.

We may also consider using a Rayleigh quotient shift after the first inverse iterations. However, once the vector has settled down to some precision, final convergence is usually rapid and another factorization would not pay.

To check convergence we calculate in each iteration the current eigenvalue approximation

$$\lambda_i^{(k+1)} = \mu + \rho^c(\bar{x}_{k+1}) \qquad k = 1,2,\ldots$$

and a relative tolerance

$$t_i^{(k+1)} = \frac{|\lambda_i^{(k+1)} - \lambda_i^{(k)}|}{\lambda_i^{(k+1)}} \quad ; \quad \lambda_i^{(1)} = 0.0$$

where $\lambda_i^{(k+1)}$ is the new approximation obtained in the k'th iteration. We say that convergence has been reached once $t_i^{(k+1)}$ is smaller than a prescribed tolerance.

Error bounds on the eigenvalues are easily obtained in the iteration. Let ℓ be the last iteration, then we evaluate

$$(A - \mu B)\, \bar{\bar{x}}_{\ell+1} = B\, \bar{x}_\ell$$

and

$$\sigma^2 = \frac{(\bar{x}_\ell, B\,\bar{x}_\ell)}{(\bar{\bar{x}}_{\ell+1}, B\bar{\bar{x}}_{\ell+1})}$$

and have with σ positive

$$\mu - \sigma \leq \lambda_i \leq \mu + \sigma$$

where μ is the current shift and λ_i is the exact eigenvalue of $Av = \lambda Bv$ [5].

Note that in practical examples $p(\mu_k)$ can be much larger than the overflow of the machine and that therefore a scale factor is used in the evaluation of the polynomial values.

In order to find the largest eigenvalues in Eq. (2.20), i.e.

$$Bv = \varkappa Av$$

the solution scheme is modified in an obvious way to iterate from the right to $\varkappa_n$. In this case we use $\mu_1 = ||B||/\lambda_{1A}$, where λ_{1A} is the smallest eigenvalue of A and $||B||$ is any convenient norm of B. We only need an approximation for λ_{1A} obtained from a few inverse iterations on A. If $\mu_1 < \varkappa_n$ we need to increase the shift until all pivots in the triangular factorization of $(B - \mu_1 A)$ are negative.

3.6 $Av = \lambda Bv$ with B Diagonal and Non-Negative Definite

We are particularly interested in the case B diagonal with some zero diagonal elements. Assume that we have t zero diagonal elements in B. Instead of Eq. (3.1) consider the equivalent problem

$$Bv = \varkappa Av \tag{3.29}$$

where $\varkappa = 1/\lambda$ and A is positive definite. By simple substitution we find that Eq. (3.29) is satisfied with the eigenvalues $\varkappa_i = 0 \quad i = 1,\ldots,t$ and the corresponding eigenvectors $v_i = e_j$ where $i = 1,\ldots,t$ and the j are selected to correspond to the zero diagonal elements in B. It follows that in $Av = \lambda Bv$ we have t infinite eigenvalues. Physically an infinite eigenvalue represents an infinite frequency which arises because zero mass has been associated with a degree of freedom.

In the algorithm developed no difficulty at all arises in finding the lowest eigenvalues and corresponding vectors in Eq. (3.1) when B has zero or small diagonal elements. If, instead, a program is used which transforms the generalized eigenvalue problem into the standard form, Eq. (3.2), it is necessary to have large enough diagonal elements to preserve numerical stability.

3.7 Programs SECANT and SECANTD

Both programs use the algorithm described in this chapter to calculate the smallest eigenvalues and associated vectors in the problem $Av = \lambda Bv$. Program SECANT was written for the case B banded and SECANTD for the case B diagonal non-negative definite. To obtain programs which calculate largest eigenvalues and corresponding vectors in the problem $Bv = \mu Av$ only a few modifications are necessary.

The programs have been established as efficient in-core solvers. During execution both matrices A and B together with the iteration vectors are in high speed storage. Therefore the maximum system size that can be analyzed is governed by the high speed storage available. As was pointed out the determinant search technique is most efficient on systems with small bandwidth, in which case on a reasonable size computer the order of the matrices can be large.

The calling parameters, storage requirements together with the program listings are given in Appendix II. Note that SECANTD is called by program MODES described in Chapter 5.

4. EIGENVALUE ANALYSIS USING A SUBSPACE ITERATION TECHNIQUE

4.1 Introduction

In the previous chapter an algorithm was developed which uses factorizations and then vector inverse iterations to find sequentially eigenvalues and corresponding vectors. If the bandwidth of the system is large, a triangular factorization is much more costly than a vector inverse iteration. In this case an algorithm which uses primarily vector iteration and only a few factorizations altogether is more economical. Such algorithm is presented in this chapter. The technique used is a subspace iteration on operators A and B simultaneously. First the theory is presented. Then various numerical problems encountered in the solution are discussed. The main difficulty is the selection of the starting subspace. Various schemes have been considered, and a very simple but effective way is now used in the programs. For the solution of the small generalized eigenvalue problem in the subspace iteration, a generalized Jacobi method, iterating directly on both operators is used.

Program SSPACE is introduced which finds the smallest eigenvalues and associated vectors, but has also been used with small modifications to calculate all eigenvalues in a specified interval. Example analyses are given in Chapter 6.

4.2 A Subspace Iteration Algorithm for the Generalized Eigenvalue Problem

We want the smallest eigenvalues and corresponding vectors of the problem

$$Av = \lambda Bv \tag{4.1}$$

when B is positive definite or diagonal non-negative definite. The basic idea is the simultaneous iteration with a number of vectors. Let X_k store p iteration vectors after k iteration steps, then we could solve

$$AX_{k+1} = B X_k R_{k+1}^{-1} \qquad k = 1,2,... \tag{4.2}$$

where R_{k+1} is an upper triangular matrix which ensures that the vectors in X_{k+1} are B-orthogonal.

We recognize Eq. (4.2) as simple inverse iteration with Gram-Schmidt orthogonalization. Provided the starting vectors in X_1 are not deficient in the eigenvectors corresponding to λ_1 to λ_p and $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda p < \lambda_{p+1}$ we have

$$X_k \to V \; ; \; R_k \to \Lambda \quad \text{as } k \to \infty$$

where

$$AV = BV\Lambda$$

and

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \cdot & & \\ & & \cdot & \\ & & & \lambda_p \end{bmatrix} \; ; \; V = \begin{bmatrix} v_1 \ldots v_p \end{bmatrix}$$

The i'th column in X_k converges to the i'th eigenvector with a rate of $\max\left(\frac{\lambda_{i-1}}{\lambda_i}, \frac{\lambda_i}{\lambda_{i+1}}\right)$. This convergence rate results from the Gram-Schmidt orthogonalization of the iteration vectors from the left to the right. We note a disadvantage of the iteration. If, for example, the first column is not very rich in v_1, but

the third column is, then in this iteration no advantage is taken of it.

We should be able to do much better than in Eq. (4.2). Assume that the starting vectors in X_1 span the p-dimensional least dominant subspace, but are not eigenvectors. A good iteration scheme would find in this case the eigenvectors in a single step. However, using Eq. (4.2) the number of iterations needed depends on how rich the individual starting vectors are in their final corresponding eigenvectors and on the convergence rates.

In order to improve upon the iteration scheme in Eq. (4.2) we need to realize that we are actually iterating with a p-dimensional subspace. Let the columns in X_k span the space denoted by $\mathcal{E}_k$, then

$$\mathcal{E}_{k+1} = \{x \mid Ax = By \ ; \ y \in \mathcal{E}_k\} \tag{4.3}$$

Furthermore, the same sequence of subspaces as by Eq. (4.2) is also generated using

$$AX_{k+1} = BX_k \tag{4.4}$$

This seems to contradict the fact that in this iteration each column in X_{k+1} is known to converge to the least dominant eigenvector. Actually there is no contradiction. Although in exact arithmetic the X_{k+1} generated by the rules in Eqs. (4.2) and (4.4) span the same subspaces, the X_{k+1} in Eq. (4.4) become a poorer and poorer basis. The Gram-Schmidt orthogonalization in Eq. (4.2) preserves numerical stability in generating an orthogonal basis in each subspace.

Our aim is to find in $\mathcal{E}_{k+1}$ a basis of vectors which are much closer to the eigenvectors sought than the columns in X_{k+1} in Eq. (4.2). The following algorithm gives an improved convergence.

For k = 1,2,... , iterate from $\mathcal{E}_k$ to $\mathcal{E}_{k+1}$

$$A\bar{X}_{k+1} = BX_k \tag{4.5}$$

Find the projections of the operators A and B onto $\mathcal{E}_{k+1}$

$$A_{k+1} = \bar{X}^T_{k+1} \; A\bar{X}_{k+1} \tag{4.6}$$

$$B_{k+1} = \bar{X}^T_{k+1} \; B\bar{X}_{k+1} \tag{4.7}$$

Solve for the eigensystem of the projected operators

$$A_{k+1} \; Q_{k+1} = B_{k+1} \; Q_{k+1} \; \Lambda_{k+1} \tag{4.8}$$

Find an improved approximation to the eigenvectors

$$X_{k+1} = \bar{X}_{k+1} \; Q_{k+1} \tag{4.9}$$

where then

$$\Lambda_k \to \Lambda \quad ; \quad X_k \to V \quad \text{as } k \to \infty$$

The better convergence rate is immediately observed by referring to the extreme case discussed with Eq. (4.2) on page 42. In this case the above algorithm yields in one step the p least dominant eigenvectors. This is obvious because we actually solve the eigenvalue problem of A and B in the least dominant p-dimensional subspace.

Rutishauser used a different subspace iteration algorithm [12], but his convergence analysis is applicable to the above scheme. It shows that the asymptotic convergence rate of the i'th vector to an eigenvector is λ_i/λ_{p+1}, where the iteration is performed with p vectors. Although this is an asymptotic

convergence rate, it indicates that the vectors corresponding to the lowest eigenvalues converge fastest. Also, a high convergence rate can be obtained by using many more vectors than there are eigenvectors required. Multiple eigenvalues do not decrease the convergence rate as long as $\lambda_p < \lambda_{p+1}$.

In practice we are much interested to know what happens already in the first few iterations. Referring to Section 2.2 we identify Eqs. (4.6) to (4.9) as a Ritz analysis with the vectors in $\bar{X}_{k+1}$ as the Ritz functions. Therefore the eigenvalues in Λ_{k+1} are stationary points in conformity with Rayleigh's minimum principle and they are upper bounds on the eigenvalues λ_1 to λ_p. Also, we recall that in a Ritz analysis the lower eigenvalues are approximated best [13].

As was pointed out already, the same sequence of subspaces is generated using either of the three iteration schemes. However, the number of operations required in one step using Eq. (4.2), Eq. (4.4), or Eqs. (4.5) to (4.9) are different. Therefore, a combination of all three schemes has been implemented.

A number of subspace iterations have been presented by different authors. Originally, in 1957, Bauer proposed a bi-iteration method for solving $Av = \lambda v$ with arbitrary matrix A [14]. Rutishauser specialized the idea to the case A symmetric [12]. Jennings introduced a simultaneous iteration method, in which he calculated a linear prediction matrix, to predict better vectors from the current iteration vectors [15][16][17].

The algorithm in Eq. (4.5) to (4.9) has the advantage that it gives a direct solution of the generalized eigenvalue problem,

and we have a clear geometric understanding of the solution procedure. In this section only theory was presented. But, in my opinion, the value of an algorithm can only be judged after exposure of all its numerical problems.

4.3 A Generalized Jacobi Iteration for the Problem $Av = \lambda Bv$

In Eq. (4.8) we calculate the eigensystem of the projected operators. The problem may be restated as

$$Av = \lambda Bv \tag{4.10}$$

where A and B are full matrices of order p and B is positive definite.

We observe two features. In the first calculation of the operator projections, B in Eq. (4.10) may be quite ill-conditioned. This can be the case in the analysis of building frames (see Section 6.5). Secondly, we note that as the iteration vectors approach the eigenvectors, A and B in Eq. (4.10) tend towards diagonal form.

If $B = SS^T$, then the problem in Eq. (4.10) is equivalent to

$$(S^{-1}AS^{-T})\ (S^Tv) = \lambda(S^Tv) \tag{4.11}$$

Efficiently, S is taken as the Cholesky factor of B, in which case $S = \tilde{L}$. If B is well-conditioned with respect to inversion, then this is a very stable process. However, when B is ill-conditioned, this process is itself ill-conditioned. We know that as B becomes semidefinite, the system has very large eigenvalues. However, as $\lambda_n \leq ||\tilde{L}^{-1}A\tilde{L}^{-T}||$ the elements in $\tilde{L}^{-1}A\tilde{L}^{-T}$ are then very large and the eigenvalues of normal size are determined inaccurately.

We may also find a matrix S using the spectral decomposition of B, i.e., $B = RD^2R^T$, in which case $S = RD$. This has an advantage if B is ill-conditioned, because the ill-conditioning of B is now concentrated in the small elements of D. In $S^{-1}AS^{-T}$ only those rows and columns corresponding to the small elements in D will have large elements, and the eigenvalues of normal size are more likely to be preserved.

A small example on which both transformations break down, although the finite eigenvalue can be well determined, is given by

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}; \quad B = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

solving

$$\det\begin{bmatrix} 2-2\lambda & 1 \\ 1 & 2 \end{bmatrix} = 0 \quad \text{gives } \lambda_1 = \frac{3}{4}; \quad (\lambda_2 = \infty)$$

In the programs developed a different solution is used which avoids a factorization of B and takes advantage of the fact that A and B tend towards diagonal form [18]. Consider the case $n = 2$ as in the standard Jacobi iteration

$$A = \begin{bmatrix} a_{jj} & a_{jk} \\ a_{kj} & a_{kk} \end{bmatrix}; \quad B = \begin{bmatrix} b_{jj} & b_{jk} \\ b_{kj} & b_{kk} \end{bmatrix} \tag{4.12}$$

Our aim is to find the two B-orthonormal vectors which also diagonalize A. The columns in V, where

$$V = \begin{bmatrix} 1 & \alpha \\ \gamma & 1 \end{bmatrix} \tag{4.13}$$

completely determine the directions of these vectors. The coefficients α and γ are obtained from the condition that the off-diagonal elements in $V^T AV$ and $V^T BV$ shall vanish . We note that in general, the diagonal elements in $V^T BV$ are not unity because the columns in V are eigenvectors which are not B-orthonormalized. The eigenvalues are the ratios of the diagonal elements in $V^T AV$ and $V^T BV$. The equations for α and γ are

$$\alpha a_{jj} + (1 + \alpha\gamma)\, a_{jk} + \gamma a_{kk} = 0 \tag{4.14}$$

$$\alpha b_{jj} + (1 + \alpha\gamma)\, b_{jk} + \gamma b_{kk} = 0 \tag{4.15}$$

If A and B are scalar multiples we set $\alpha = 0$ and obtain $\gamma = \frac{-a_{jk}}{a_{kk}}$. In the general case let

$$\bar{a}_{kk} = a_{kk}\, b_{jk} - b_{kk}\, a_{jk}$$

$$\bar{a}_{jj} = a_{jj}\, b_{jk} - b_{jj}\, a_{jk}$$

$$a = a_{jj}\, b_{kk} - a_{kk}\, b_{jj}$$

$$\alpha = \frac{\bar{a}_{kk}}{x} \quad ; \quad \gamma = -\frac{\bar{a}_{jj}}{x}$$

then

$$x^2 - a\,x - \bar{a}_{kk}\,\bar{a}_{jj} = 0$$

$$x_{1,2} = \frac{a}{2} \pm \sqrt{\frac{a^2 + 4\bar{a}_{kk}\,\bar{a}_{jj}}{4}}$$

where we use the absolutely larger value of x.

In general, we know that this congruence transformation is possible provided one of the operators is positive definite. Note

the corresponding analogy between Eqs. (4.14) and (4.15).

Let us use the method to solve the small example above. In this case

$$V = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{bmatrix}$$

and

$$V^T A V = \begin{bmatrix} \frac{3}{2} & 0 \\ 0 & 2 \end{bmatrix} ; \quad V^T B V = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

hence

$$\lambda_1 = \frac{3}{4} ; \quad \lambda_2 = \infty$$

The eigenvalue problem $Av = \lambda Bv$ with matrices of order larger than two can now be solved in a similar way as the standard eigenvalue problem when rotation matrices are used. For example, we may reduce A and B simultaneously to tridiagonal form and then use a determinant search technique to find the eigenvalues.

In the subspace iteration programs the method is used in form of a Jacobi iteration on A and B simultaneously. This way we take advantage of the small off-diagonal elements in A and B as the number of subspace iterations increases.

The Jacobi iteration on A and B can be implemented in various ways. One may simply zero the off-diagonal elements sequentially, irrespective of how small they are already. But it is more efficient to use a threshold iteration, in which the elements are only zeroed, if they are in magnitude larger than the current threshold.

Physically, in the diagonalization we want to reduce the coupling between the generalized degrees of freedom i and j. A measure of the coupling is given by the coupling factors $(a_{ij}^2/a_{ii}\ a_{jj})^{\frac{1}{2}}$ and $(b_{ij}^2\ b_{ii}\ b_{jj})^{\frac{1}{2}}$. The iteration is most efficient if we annihilate first the most significant and then the smaller coupling, until the diagonal forms have been reached. The following natural scheme is used in the programs:

(i) Initialize the threshold for the k'th sweep

(ii) For all (i,j) with $i < j$ calculate the coupling factors and apply a transformation if either is larger than the threshold.

(iii) Calculate new eigenvalue estimates.

(iv) Compare the new eigenvalue estimates with those from the previous sweep. If the change is too large, start the next sweep. Before convergence is accepted, check if all coupling factors are smaller than the tolerance on the eigenvalues.

In (i) we need to make a decision what threshold should be used. If we want to have the coupling factors smaller than 10^{-12} after about six iterations, it is reasonable to use as the threshold 10^{-2k}. In (iv) it may be that the relative change in the eigenvalue estimates are smaller than 10^{-12}, but the iteration continues until the coupling factors are also small enough. Note, in (ii) we only need the square of the coupling factors and compare it with the square of the threshold tolerance, but this still requires six multiplications at each check. We could use

(a_{ij}/a_m) and (b_{ij}/a_m), where

$a_m = \frac{1}{p}\sum_i a_{ii}$, $b_m = \frac{1}{p}\sum_i b_{ii}$ and would only need two multiplications at each check. However, in example analyses, convergence was not reached as fast. It is important to use the coupling factors as thresholds because the diagonal elements in A and B, calculated in the subspace iterations can vary significantly. For example, in the subspace iteration analysis of the frame in Section 6.5, the largest ratio of the diagonal elements in B is in each iteration about 10^4 or more.

Considering the numerical stability of the method, we realize that V is not an orthogonal matrix. Therefore, element growth could occur in A and B. This element growth has never been observed but would have to be counteracted by normalizing the elements in B after each sweep.

Note that if the method is used on the standard eigenvalue problem, we find $\alpha = -\gamma$ and recognize V as a multiple of the Jacobi rotation matrix, where

$$\cos\theta = \frac{1}{\sqrt{1+\gamma^2}}, \quad \sin\theta = \frac{\gamma}{\sqrt{1+\gamma^2}}$$

4.4 Selection of Initial Transformation Vectors

The choice of the initial transformation vectors in X_1 is most important. If these vectors are already close to eigenvectors, convergence can be obtained in a few iterations. On the other hand, when the starting subspace spanned by the vectors is only a poor approximation to the least dominant p-dimensional

subspace, many iterations are required and the algorithm can become expensive.

Therefore, whenever we have starting vectors which quite well approximate the eigenvectors sought, these vectors should be used in X_1. In this case, the algorithm is ideally suited for solution. In practice, this may arise for instance in dynamic optimization. As the structure is modified in small steps, the eigensystem of the previous structure would be a good approximation to the eigensystem of the new structure. Sometimes it may be difficult to judge if good transformation vectors are known. But the conventional Ritz analysis in which load patterns are specified (see Section 2.2) the component mode synthesis and related methods summarized by Uhrig, can all be a 'good' first subspace iteration [3].

Assume that we do not have by previous experience or analysis 'good' transformation vectors. Consider the problem of choosing the starting subspace using the operators A and B only, i.e. the columns in X_1 shall be established by the computer using only the elements in A and B.

One way of solution would be to find X_1 from two operators A' and B' which have a least dominant p-dimensional subspace close to the one of A and B. Naturally, we would like to find A' and B' and its eigensystem with little effort. Different possibilities can be investigated.

In the component mode synthesis we establish the vectors in X_1 from eigenvectors of a number of pairs of smaller operators

A' and B' which are related to A and B. If all these vectors have to be solved for, as we now assume, then this procedure is very expensive. Furthermore, it is difficult if not impossible to have the computer establish adequate pairs of operators A' and B'.

A different scheme could be used, in which we reduce the order of the eigenvalue problem. The operator B' is obtained by lumping the elements in B into q diagonal positions, where $p < q \ll n$. The other coordinates are eliminated and the q-dimensional eigenvalue problem is solved. This procedure was described in Section 2.2 as static condensation of the massless degrees of freedom. It was pointed out that it is a subspace iteration on the operators A and B', where the vectors in $B X_1$ of Eq. (4.5) are unit vectors with entries at the q-coordinates. We should expect to obtain a better solution using the same starting transformation vectors but performing a subspace iteration on the operators A and B.

Suitable operators A' and B' may also be obtained by reducing the bandwidths of A and B. If the bandwidths can be reduced to very small, then the solution for the transformation vectors is economical. Assume that B is diagonal, so that we are only concerned with the reduction of the bandwidth in A. Physically, a bandwidth reduction means a lumping of stiffness, which can hardly be carried out adequately without the analyst specifying it. Otherwise a model may be analyzed which in no way resembles any more the original model of the operators A and B.

It is more direct to start immediately iterating with the operators A and B. The following seemed to be a good scheme. We use inverse iteration starting with a vector having +1 at each coordinate to find the first transformation vector. We repeat the same iteration to find all other transformation vectors, but the iteration vector is now orthogonalized to the transformation vectors already accepted. Each time the iteration is stopped once the eigenvalue estimate changes by less than a tolerance. The defect of this scheme is the following. If the tolerance for accepting an iteration vector is very high, then we virtually solve the eigenvalue problem in an expensive way. On the other hand, if the tolerance is low, then the iteration vector may be an approximation to an eigenvector which we do not want to find. Therefore, the starting subspace is not 'good' and may even be orthogonal to one of the required eigenvectors. Also, we need at least three to four inverse iterations for each transformation vector. In terms of operations this is approximately equivalent to three subspace iterations.

The final conclusion is that it is best to start directly with subspace iterations on the operators A and B. The following vectors in $R = B X_1$, which represent the right hand side in Eq. (4.5), have been found most effective. The first vector in R is simply the diagonal of B. The other vectors are unit vectors with +1 at a coordinate with a large ratio b_{ii}/a_{ii}. The algorithm below describes how R is established.

```
C
C      ESTABLISH STARTING TRANSFORMATION VECTORS
         DO 200 I=1,N
 200     R(I,1)=B(I)
         IF (NC.EQ.1) GO TO 295
         ND=N/NC
         DO 210 I=1,N
 210     W(I)=B(I)/A(I)
         DO 220 I=1,N
         DO 220 J=2,NC
 220     R(I,J)=0.0
C
         L=N-ND
         DO 240 J=2,NC
         RT=0.0
         DO 260 I=1,L
         IF (W(I).LT.RT) GO TO 260
         RT=W(I)
         IJ=I
 260     CONTINUE
         DO 280 I=L,N
         IF (W(I).LE.RT) GO TO 280
         RT=W(I)
         IJ=I
 280     CONTINUE
         W(IJ)=0.0
         L=L-ND
 240     R(IJ,J)=1.
C
 295     CONTINUE
```

In the algorithm N = order of matrices, NC = number of transformation vectors and A and B are stored as one-dimensional arrays.

The physical reasoning for the selection of these starting or load vectors is as follows. First of all, not to miss a mode all mass degrees of freedom are excited in the first vector. The other vectors must be linearly independent and should excite points of maximum mass and flexibility. Also, for better convergence the unit entries in the second to last vector should not be clustered together very much. These ideas are also used in the block iteration solution (see Chapter 5), but the algorithm used in that case (normally) reads A and B only once into core.

Observe that in exact arithmetic, we may replace a vector by any linear combination of the others and itself and we still have the same starting subspace.

Referring to Section 2.2 we need to note how closely related this first subspace iteration is to a static condensation analysis. Assume that we would use only unit vectors and that we had lumped the mass at the coordinates at which the unit loads are applied, then the static condensation analysis would give the same results as the first subspace iteration.

4.5 Numerical Aspects

There are various numerical questions which needed a thorough investigation in the implementation of the subspace iterations.

4.5.1 Dimension of Subspace

As mentioned already there is advantage in using more iteration vectors than there are eigenvectors required. The more vectors we take, the larger the initial subspace we span and the higher the ultimate convergence rate of an iteration vector. Therefore, we iterate with q vectors when we want to find the p least dominant eigenvectors, $q > p$. It is most important in the iteration that we obtain monotonic convergence of p vectors to the eigenvectors sought. Assume that our starting subspace is almost orthogonal to one of the p eigenvectors. Then for many iterations we may have no approximation to that eigenvector. But suddenly it appears and rapid convergence to the required eigenvector is obtained.

A reasonable number of iteration vectors is given by

$$q = \min\{2p, p+8\} \tag{4.16}$$

With this formula we do not use an excessive number of iteration vectors, we allow for multiple roots and the dimension of the subspace is large enough to expect monotonic convergence.

4.5.2 Convergence

An advantage of the subspace iteration over vector inverse iteration with shifting followed by matrix deflation is that high precision in the eigenvalues and vectors is not required for numerical stability (see Section 3.2). In practice, we are probably satisfied with 5 to 6 digit accuracy in the eigenvalues.

In the iteration new eigenvalue approximations are only obtained when the projections of the operators are calculated. Assume that in the iterations (k-1) and k we calculate eigenvalue approximations $\lambda_i^{(k)}$ and $\lambda_i^{(k+1)}$ respectively, then we also find

$$t_i^{(k+1)} = \frac{\left|\lambda_i^{(k+1)} - \lambda_i^{(k)}\right|}{\lambda_i^{(k)}} \; ; \; \lambda_i^{(1)} = 0.0 \tag{4.17}$$

All those eigenvalues for which $t_i^{(k+1)}$ is smaller than a prescribed value, which we call RTOL, are said to have converged.

The subspace iteration is stopped once the required p eigenvalues have converged or the maximum number of iterations, NITEM, has been carried out. NITEM naturally must depend on the eigenvalue accuracy asked for. The maximum which RTOL should probably take is 10^{-6} and then we may use NITEM = 12.

4.5.3 Check Calculations

The starting subspace described in Section 4.4 has been very satisfactory for obtaining monotonic convergence. In the example analyses used always all required eigenvalues and vectors were approximated monotonically with increasing accuracy. A different starting subspace spanned by only unit coordinate vectors was not as 'good'. It happened in some analyses that a required eigenvalue was missing.

In general we can make sure that we found the lowest p eigenvalues using the eigenvalue separation theorem (see Section 3.3.2). A factorization must be carried out at a shift to the right of the p'th eigenvalue obtained in the iteration. In order to know where the check can be applied it is necessary to establish bounds for the 'exact' eigenvalues λ_i.

Let $\lambda_i^{(\ell+1)}$ be the eigenvalue approximation and $v_i^{(\ell+1)}$ be the orthonormalized vector obtained in the last iteration. Assume that B is positive definite, then we have

$$r_i = (A - \lambda_i^{(\ell+1)} B)\, v_i^{(\ell+1)} \quad ; \quad \eta_i = (r_i, B^{-1} r_i)^{\frac{1}{2}}$$

and

$$\lambda_i^{(\ell+1)} - \eta_i \leq \lambda_i \leq \lambda_i^{(\ell+1)} + \eta_i \qquad (4.18)$$

If B is diagonal non-negative definite we could consider the problem $Bv = \varkappa Av$ instead (see Section 3.6). These error bound calculations can be rather expensive, and in practice we probably don't need exact error bounds. For the check we may also use a conservative estimate for a region in which λ_i lies, given by

$$\lambda_i^{(\ell+1)}(1 - 10^{-2}) < \lambda_i < \lambda_i^{(\ell+1)}(1 + 10^{-2}) \qquad (4.19)$$

where only the lowest eigenvalues which all have reached convergence should be included.

It is now necessary to evaluate upper bounds on eigenvalue clusters (Fig. 5). A check can be applied at any one of these bounds. Assume that in Fig. 5 at E one eigenvalue is missing but none at F. Then we could use the interval from F to E in another subspace iteration in which all eigenvalues in this interval are calculated.

4.5.4 Shifting

The lowest eigenvalues converge first in the iteration. It seems reasonable to try and shift after the first iterations in order to speed convergence of the higher eigenvalues. The main danger is to shift too far to the right and lose convergence to the lower eigenvalues. On the other hand a very small and therefore conservative shift does not give a significant increase in the convergence speed of the higher eigenvalues. A reliable and good shift can only be determined once the eigenvalue spectrum is known approximately. All required eigenvalues need have settled down before shifting. But then convergence is reached in the next subspace iterations and an additional factorization is uneconomical.

4.5.5 Operation Saving

Once the lowest p_ℓ, $p_\ell < p$, eigenvalues have converged, we may like to stop iterating on the corresponding vectors. This

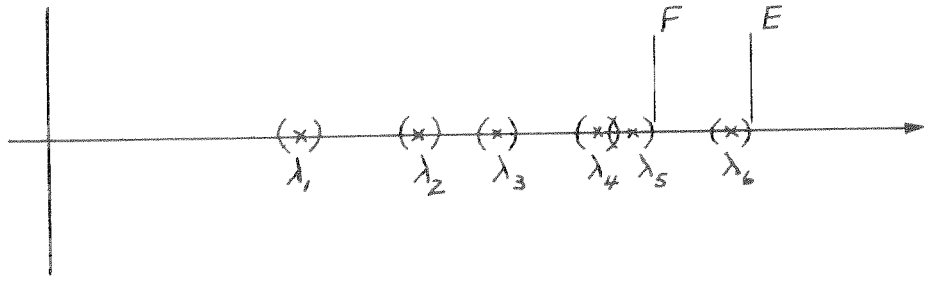


FIGURE 5: BOUNDS ON EIGENVALUES TO APPLY STURM SEQUENCE CHECK

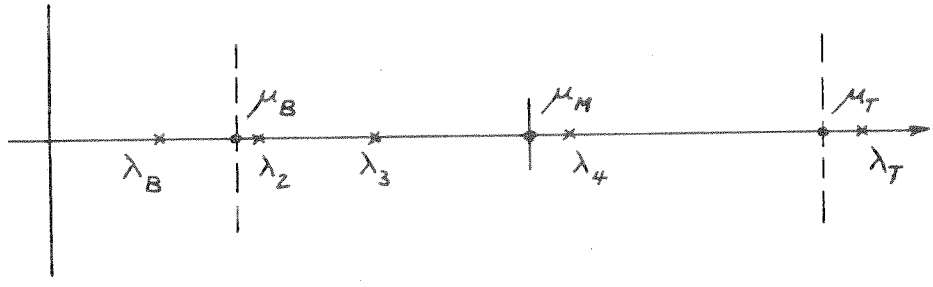


FIGURE 6: CALCULATION OF EIGENVALUES IN THE INTERVAL μ_B TO μ_T

way we would save the inverse iteration on p_ℓ vectors and we would not include them in the calculation of the new right hand side in Eq. (4.9). However, this can only be done once the space spanned by the rest of the vectors is orthogonal to the space spanned by the p_ℓ vectors. Numerically, this means that the off-diagonal elements in the p_ℓ first columns of B_{k+1} in Eq. (4.7) must be very small. In general, convergence for all p eigenvalues to RTOL=10^{-6} is obtained before this condition is reached.

4.6 Operation Count

The triangular decomposition of A before the subspace iteration is started requires $(\frac{1}{2} n m_A^2 + \frac{3}{2} n m_A)$ operations. Recall that one operation equals one multiplication which nearly always is followed by an addition, and that m_A and m_B are assumed to be full.

We now need to distinguish between three iteration schemes (a) simple inverse iteration of all vectors, (b) inverse iteration followed by Gram-Schmidt orthogonalization and (c) inverse iteration followed by the calculation of the projections. In the first two iteration methods we do not calculate eigenvalue estimates. The simple inverse iteration is used only once or twice and we do not normalize the iteration vectors. Table 1 summarizes the steps in each iteration and the corresponding number of operations. Terms involving the bandwidths only have been neglected.

We note that for $m \gg q$ we need in all three schemes about the same number of operations. But when A and B have small or

TABLE 1: Operation count for subspace iterations a, b and c.

METHOD	CALCULATION	NUMBER OF OPERATIONS	
		$m = m_A = m_B$	$m = m_A; m_B = 0$
(a) Simple Inverse Iteration	$A\bar{X}_{k+1} = Y_k$	$nq(2m+1)$	$nq(2m+1)$
	$Y_{k+1} = B\bar{X}_{k+1}$	$nq(2m+1)$	nq
	TOTAL	$2nq(2m+1)$	$2nq(m+1)$
(b) Inverse Iteration with Gram-Schmidt Orthogonal-ization	$A\bar{X}_{k+1} = Y_k$	$nq(2m+1)$	$nq(2m+1)$
	$\bar{Y}_{k+1} = B\bar{X}_{k+1}$	$nq(2m+1)$	nq
	$Y_{k+1} = \bar{Y}_{k+1} R^{-1}_{k+1}$	$\frac{nq}{2}(3q+3)$	$\frac{nq}{2}(3q+3)$
	TOTAL	$2nq(2m+\frac{3}{4}q+\frac{7}{4})$	$2nq(m+\frac{3}{4}q+\frac{7}{4})$
(c) Iteration Using Projection of Operators	$A\bar{X}_{k+1} = Y_k$	$nq(2m+1)$	$nq(2m+1)$
	$A_{k+1} = \bar{X}^T_{k+1} Y_k$	$\frac{nq}{2}(q+1)$	$\frac{nq}{2}(q+1)$
	$\bar{Y}_{k+1} = B\bar{X}_{k+1}$	$nq(2m+1)$	nq
	$B_{k+1} = \bar{X}^T_{k+1} \bar{Y}_{k+1}$	$\frac{nq}{2}(q+1)$	$\frac{nq}{2}(q+1)$
	$A_{k+1} Q_{k+1} = B_{k+1} Q_{k+1} \Lambda_{k+1}$	$0(q^3)$	neglected ...
	$Y_{k+1} = \bar{Y}_{k+1} Q_{k+1}$	nq^2	nq^2
	TOTAL	$2nq(2m+q+\frac{3}{2})$	$2nq(m+q+\frac{3}{2})$

medium bandwidth a combination of a, b and c can be significantly more economical. In program SSPACE (see Section 4.8) we perform first subspace iterations using a and b and then we iterate with c to obtain eigenvalue estimates.

Consider first the case B diagonal. To give some meaning to the number of operations required in an eigenvalue solution, we define variables α and α_T. Let α be the number of factorizations equivalent in operations to a subspace iteration c, hence

$$\alpha = \frac{4qm + 4q^2 + 6q}{m^2 + 3m} \tag{4.20}$$

We note that α is independent of n. Let α_T be the total number of operations in terms of factorizations required for the subspace iterations,

$$\alpha_T = \alpha \times (\text{total no. of iterations for convergence}) \tag{4.21}$$

Assume that we use Eq. (4.16) for the relation between p and q, and that we want about five digit accuracy in the eigenvalues. Then, by experience, we need approximately 8 subspace iterations. Figures 7 and 8 show for this case the relation between α_T and m for different values of p. The same information is also given for the case $m_B = m_A$. In this comparison of Central Processor operations it is assumed that we either perform an in-core solution, or that arrays are transferred in large blocks. In this case the CP time spent in tape reading and writing is negligible [16].

We may now say that the complete eigenvalue solution involves the initial factorization of A, the factorizations given in Figs. 7 and 8, which account in terms of operations for the subspace

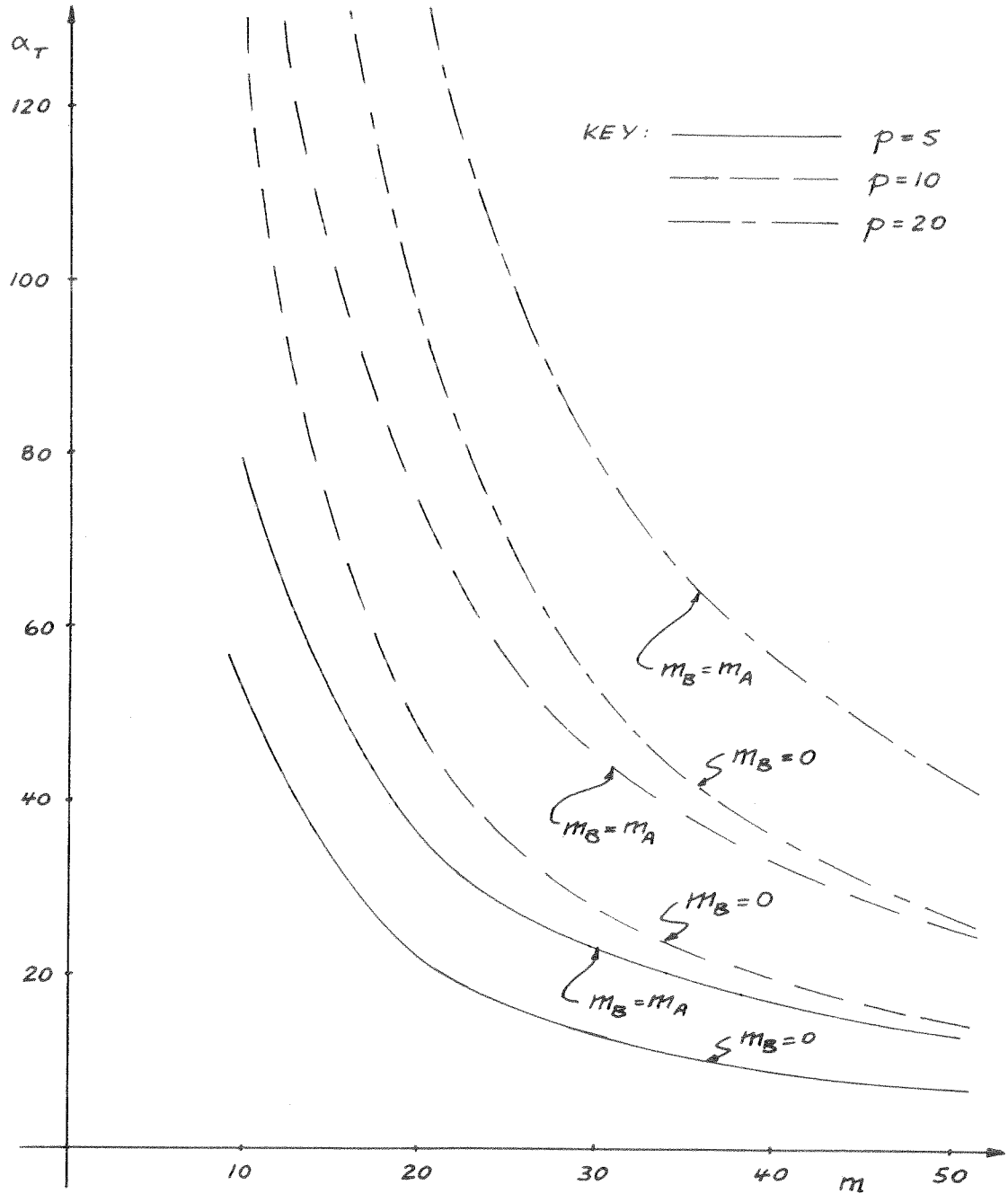


FIGURE 7: RELATION BETWEEN α_T AND m FOR p = 5, 10 AND 20.

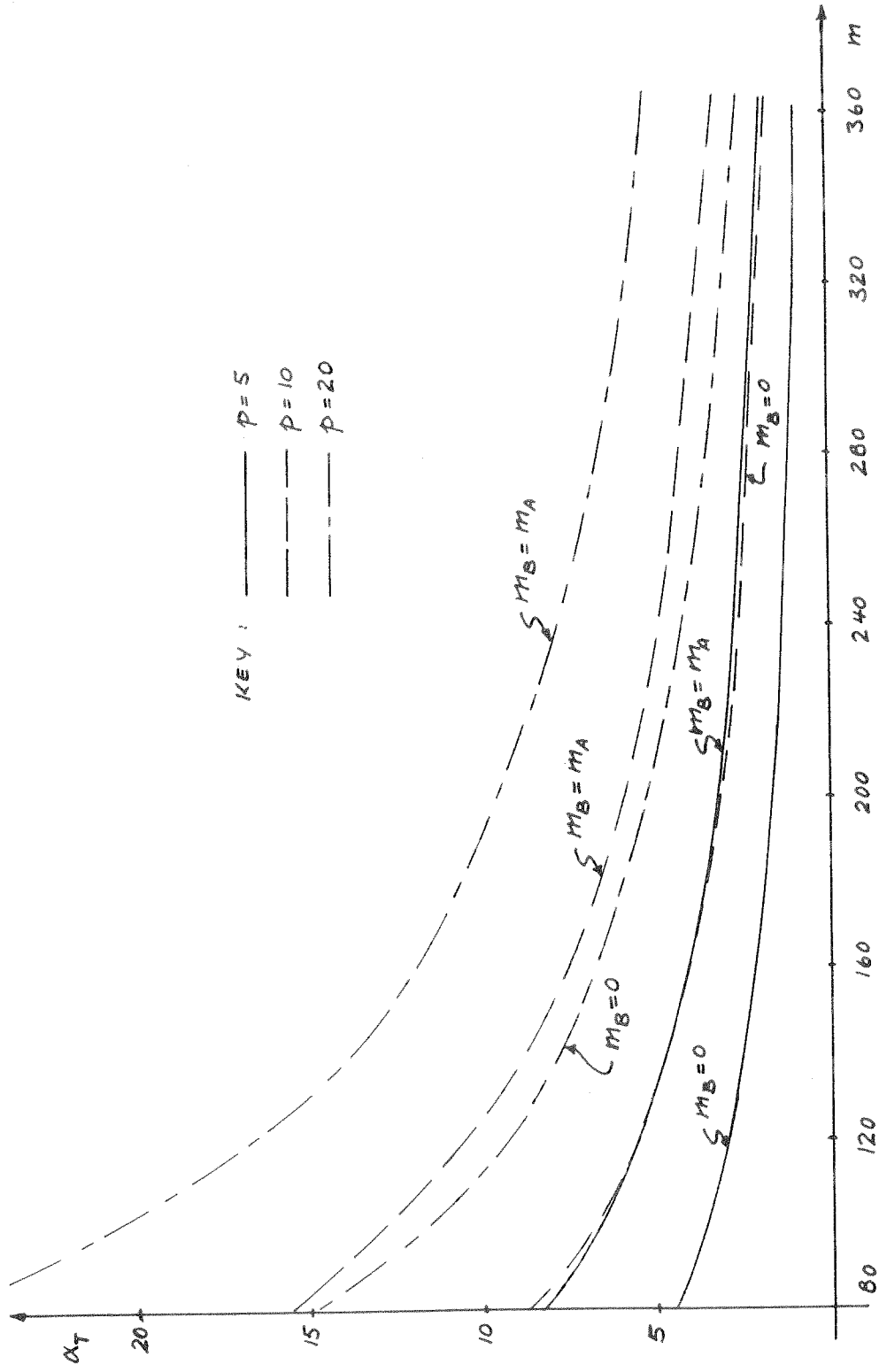


FIGURE 8: RELATION BETWEEN α_T AND m FOR P = 5, 10 AND 20

iterations and the final check calculations. When k subspace iterations are needed instead of eight, we merely use the information in Figs. 7 and 8 times k/8. This will be the case when we have already excellent starting vectors from a previous solution, or when we want the eigenvalues to full machine precision.

The operation count shows that with larger m about twice as many operations are required when B is banded. We also note that when m is large the operations required in the subspace iterations are of the order of a factorization. But when m is small the eigenvalue solution is equivalent to many factorizations. A good figure to remember is that with m = 320, q = 10 and $m_B = 0$ the subspace iterations are about equivalent to one factorization. An approximate formula for m being large and $m_B = 0$ is $\alpha_T = \frac{32}{m}q$, where we observe that for m large the only significant operations are the vector inverse iterations.

Using the determinant search technique we may require an average amount of operations equivalent to about 6 to 8 factorizations in order to calculate each required eigenvalue. A large system with a small bandwidth has probably a bandwidth between 40 and 60. We note that a subspace iteration solution is then already cheaper but not excessively.

4.7 Calculation of Eigenvalues and Vectors in an Interval

The subspace iteration algorithm was implemented to find all eigenvalues in an iterval μ_B to μ_T (Fig. 6). We use the eigenvalue separation theorem at μ_T and μ_B to find how many eigenvalues and

corresponding vectors need be calculated. Then we shift to the midpoint of the interval μ_M and start the iteration.

In the algorithm we iterate with p vectors when we want to find p eigenvalues between μ_B and μ_T. The ultimate convergence rate of the i'th iteration vector is $\max \left\{ \left| \frac{\lambda_i - \mu_M}{\lambda_B - \mu_M} \right| , \left| \frac{\lambda_i - \mu_M}{\lambda_T - \mu_M} \right| \right\}$, where λ_B and λ_T are the eigenvalues at bottom and top end next to the interval. We note that convergence can be slow for eigenvalues near the upper and lower end of the interval. To speed convergence for those eigenvalues we can wait until an eigenvalue has settled down. We then shift to it and use inverse iteration on the corresponding vector to obtain the eigenvalue to high precision. This means that we need more factorizations than in the iteration for the lowest eigenvalues, and in general this iteration is more expensive.

Apart from the slower convergence there are additional numerical difficulties in this solution.

Although rare it can happen that an iteration vector is oscillating in the space which is orthogonal to the space spanned by the p eigenvectors sought. Because the operator $(A - \mu_M B)$ is indefinite, the oscillating iteration vector can give a Rayleigh quotient which lies in the interval considered. However, this is detected when we calculate error bounds using Eq. (4.18).

Assume that we want to find p eigenvalues and that μ_M is an eigenvalue of multiplicity s, where $p > s$. As the shift is on

the eigenvalue, each of the iteration vectors will converge immediately to an eigenvector in the s-dimensional subspace and the iteration will fail. This will seldom occur and we can detect it because the last s pivot elements in the triangular factorization will be very small. But μ_M may be near an eigenvalue and the iteration process will have difficulty to converge.

It may happen that μ_B or μ_T is an eigenvalue, then we do not need to iterate for it in the subspace iteration.

Assume that we want to calculate the smallest eigenvalues from zero to a cut-off. Then we may find how many eigenvalues are smaller than the cut-off, say p, and use the subspace iteration with q vectors and shift equal to zero.

It is obvious how to calculate the largest eigenvalues in $Bv = \varkappa Av$. We merely need to shift to $\mu_1 = \|B\|/\lambda_{1A}$, where λ_{1A} is the smallest eigenvalue of A and $\|B\|$ is any norm of B (see Section 3.5).

4.8 Program SSPACE

Program SSPACE uses the subspace iteration technique presented to evaluate the smallest eigenvalues and associated vectors in the problem $Av = \lambda Bv$ when B is banded or diagonal non-negative definite. During execution both matrices A and B together with the iteration vectors are in high speed storage. The program was used for the study of the algorithm. It is written flexibly so that any combination of the three different subspace iteration schemes in Table 1 can be used. A few modifications are necessary to obtain a program which finds eigenvalues and corresponding

vectors in an interval or to calculate the largest eigenvalues in the problem $Bv = \mu Av$.

The calling parameters, storage requirements and a listing of the program are given in Appendix II.

5. PROGRAM MODES

5.1 Introduction

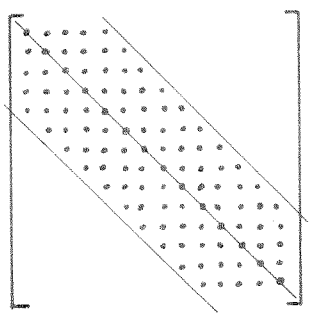
The programs presented so far all have a capacity restriction. In the solution matrices A and B together with iteration vector arrays need to fit into high speed storage. If much high speed storage is available, large systems can be analyzed. Computers have been growing continuously, but currently, even when a large computer is available, there are systems to be analyzed which are too large for an in-core solution. In general, for structural engineering problems the lowest eigenvalues and vectors of large systems are required.

In this chapter a program is presented which evaluates the lowest eigenvalues and vectors of systems that may or may not be solved in high speed storage. Program MODES solves the eigenvalue problem for B diagonal and non-negative definite. This is the most common requirement in structural enginering. The program has been used with SAP [19], but could easily be coupled to any other program which generates the matrices A and B.

5.2 Program Operation

It is assumed that matrices A and B are stored on tapes in block form (Fig. 9).

If only one block is used, program SECANTD is called for an in-core solution. In this case, it is assumed that we have either a large system with small bandwidth and SECANTD is appropriately used, or we have a small system with larger bandwidth. Then the

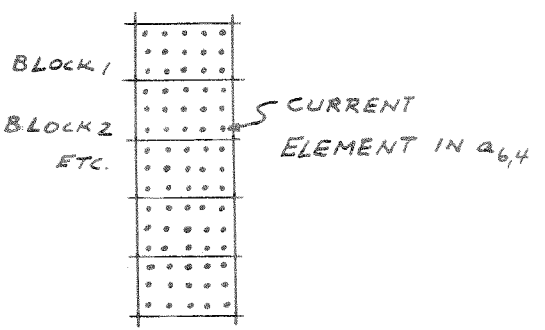


(a) MATRIX A

NEQ = 14 ; MBAND = $m_A + 1 = 5$

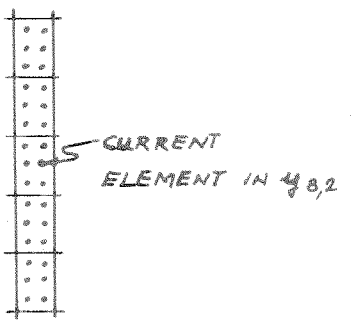
NBLOCK = NO OF BLOCKS = 5

NEQB = NO OF EQNS PER BLOCK = 3



(b) STORAGE FOR MATRIX ELEMENTS

INITIALLY UPPER BAND OF MATRIX A ; FINALLY ELEMENTS OF D AND L^T WHERE $A = LDL^T$



(c) STORAGE FOR ELEMENTS IN VECTORS

FIGURE 9 : STORAGE OF MATRIX AND ITERATION VECTORS

eigenvalue solution is still not very costly and we have the advantage of getting the eigenvalues one-by-one to high precision with error bounds.

When more blocks than one are used program SSPACEB is called and a subspace iteration solution is carried out. In each subspace iteration, the projections of A and B are calculated. As was indicated in Section 4.6 little more operations are required using this subspace iteration instead of inverse iteration followed by Gram-Schmidt orthogonalization. The advantage is that we obtain eigenvalue estimates in each iteration. Also, in a Gram-Schmidt orthogonalization much tape handling would be necessary. At convergence a Sturm Sequence check can be performed to ensure that all the required eigenvalues have been calculated.

For best program efficiency the algorithm for the factorization of A and for the vector iterations need be optimized. For details I shall refer to the program in Appendix II, but the main operations are outlined on a small example in the next sections.

5.3 Block Factorization of A

Matrix A is factored into LDL^T using simple Gaussian elimination (Eq. (3.10) with $\mu_k = 0.0$). Naturally, we work only on the upper band of A (Fig. 9). Let initially element (i,j) of the upper band of matrix A be stored in $a_{i,j}$. In the algorithm the elements in $a_{i,j}$ are changed until finally $a_{i,1} = d_{ii}$ and $a_{i,j} = \ell_{j+i-1,i}$ where $i = 1,\ldots,n$ and $j = 2,\ldots,m_A+1$. We say that a block has been reduced if all ℓ_{km} to be stored in the

block have been calculated and the associated row operations have been performed.

The only aspect which shall be presented here is the organization using blocks. It is easiest to demonstrate the algorithm by considering a small example. Figure 9 shows matrix A and its storage in the algorithm, where NEQ=14, MBAND=5, NBLOCK=5 and NEQB=3. The program will have two blocks in high speed storage. There are NBLOCK main steps in each of which one block is reduced. Consider the reduction of the first block which is typical.

In this step block 1 is kept in high speed storage. We note that its reduction affects two more blocks. The reduction is carried out in the following three operations.

(i) First block 1 is reduced as far as it does not affect another block. In the example we evaluate

$$a_{2,j} = a_{2,j} - \frac{a_{1,2}}{a_{1,1}} a_{1,j+1} \qquad (j=1,\ldots,4)$$

$$a_{1,2} = \frac{a_{1,2}}{a_{1,1}}$$

where we take the current elements in $a_{i,j}$ to evaluate new elements in $a_{i,j}$. Also

$$a_{3,j} = a_{3,j} - \frac{a_{1,3}}{a_{1,1}} a_{1,j+2} \qquad (j=1,\ldots,3)$$

$$a_{1,3} = \frac{a_{1,3}}{a_{1,1}}$$

$$a_{3,j} = a_{3,j} - \frac{a_{2,2}}{a_{2,1}} a_{2,j+1} \qquad (j=1,\ldots,4)$$

$$a_{2,2} = \frac{a_{2,2}}{a_{2,1}}$$

(ii) Block 2 is now called into core and we evaluate

$$a_{4,j} = a_{4,j} - \frac{a_{1,4}}{a_{1,1}} a_{1,j+3} \qquad (j=1,2)$$

$$a_{1,4} = \frac{a_{1,4}}{a_{1,1}} \qquad \text{etc.}$$

We change the elements in block 2 and finally have stored multipliers in $a_{1,4}$, $a_{1,5}$, $a_{2,3}$, $a_{2,4}$, $a_{2,5}$, $a_{3,2}$, $a_{3,3}$ and $a_{3,4}$.

(iii) Block 2 is stored on temporary tape and block 3 is called into its storage area. The elements in block 3 are changed as in (ii) and the block is stored on temporary tape behind block 2.

Block 1 has been reduced and is stored on tape. We call block 2 into its storage area and carry out its reduction analogously.

In (i) we calculate the bandwidth at each row and we take account of the variable bandwidth in (i), (ii) and (iii). Also, a change in the elements of a row is skipped if the multiplier ℓ_{ij} is zero.

5.4 Block Vector Iteration

Consider Eq. (4.5) which can be written as

$$LDL^T X = Y$$

where $X = \bar{X}_{k+1}$ and $Y = BX_k$. We first reduce the vectors in Y to obtain

$$L^T X = D^{-1} L^{-1} Y$$

and then solve for X by a back-substitution.

In the vector reduction and back-substitution we keep in high speed storage one reduced block of A and as many blocks of the vectors as are effected simultaneously. Let initially the (i,j) element of Y be stored in $y_{i,j}$. In the reduction and back-substitution $y_{i,j}$ is changed until finally the (i,j) element of X is stored in it.

There are again NBLOCK main steps each in the reduction and in the back-substitution. The tape organization is analogous. Referring to the example we consider the first step in the reduction which is typical.

Assume that we iterate with two vectors (Fig. 9). The first block of D and L^T now stored in elements $a_{i,j}$ and the first three blocks of the vectors are taken into high speed storage. The reduction of the vectors with the multipliers of block 1 is

$$y_{j,k} = y_{j,k} - a_{1,j}\, y_{1,k} \qquad (j=2,\ldots 5)$$

$$y_{1,k} = y_{1,k}/a_{1,1} \qquad (k=1,2)$$

where we use the current elements in $y_{j,k}$ to evaluate new elements in $y_{j,k}$. Also

$$y_{j+1,k} = y_{j+1,k} - a_{2,j}\, y_{2,k} \qquad (j=2,\ldots 5)$$

$$y_{2,k} = y_{2,k}/a_{2,1} \qquad (k=1,2)$$

$$y_{j+2,k} = y_{j+2,k} - a_{3,j}\, y_{3,k} \qquad (j=2,\ldots 5)$$

$$y_{3,k} = y_{3,k}/a_{3,1} \qquad (k=1,2)$$

Next the second block of D and L^T is taken into core. The first block of vectors is stored on temporary tape, and the vector blocks are shifted up by one. Vector block 4 is then read into the storage area which was occupied by block 3 and the reduction of the vectors continues.

In the back-substitution also always three vector blocks are in high speed storage, so that the final results in one vector block are calculated in one step.

6. EXAMPLE ANALYSES

6.1 Introduction

The aim in this chapter is to show the main characteristics of program solutions in the analysis of different kinds of structures. These include a box girder, a dam, a plane frame and a three dimensional building frame. All of these structures could have been larger, but then more computer time would have been used in the analyses without much changing the solution characteristics. The programs used are given in Appendix II.

In the subspace iteration solutions the starting subspace was generated as described in Section 4.4. Convergence was reached for RTOL = 10^{-6} (see Section 4.5.2). The total time used in the subspace iteration solution always includes the initial factorization of matrix A and the final Sturm Sequence check. Referring to Section 4.6 the sequence of subspace iterations performed in SSPACE was a,b,a,b,c,c,c,.... Program SSPACEB is called from MODES and performs only iterations c.

In program SECANT and SECANTD convergence was reached to full word precision. Reference is made to program BANEIG II, which is the latest version of BANEIG [7]. All analyses were carried out using the CDC 6400 on the Berkeley Campus with a maximum storage of 130000_8 and a 48 bit mantissa in floating point arithmetic.

6.2 Analysis of Cantilever Box

The finite element idealization of the box is shown in

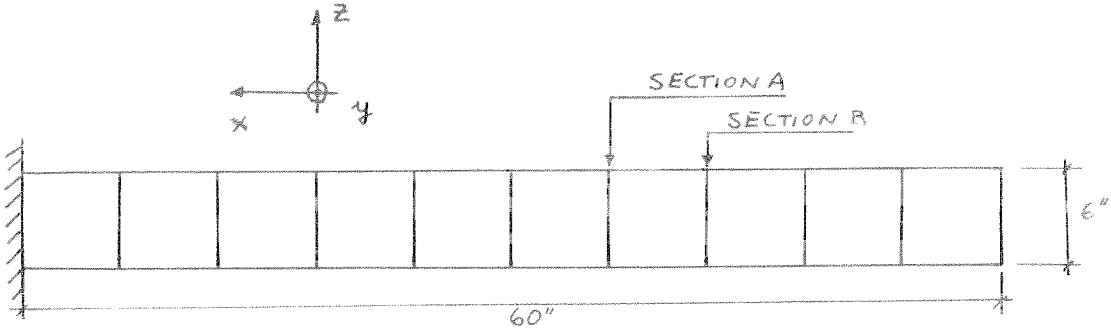
Fig. 10. The segments of the box had originally 28 degrees of freedom, but the 8 rotational degrees of freedom were condensed out before assemblage. A lumped mass formulation was used in the analysis. The order of the stiffness was 100 and the maximum half bandwidth 19.

The smallest eigenvalues of the system together with the solution times taken by the different programs are given in Table 2. For SECANTD the total time used is just the sum of the operation times for each eigenvalue. In BANEIG II the eigenvectors of the deflated system need still be transformed and this time is added.

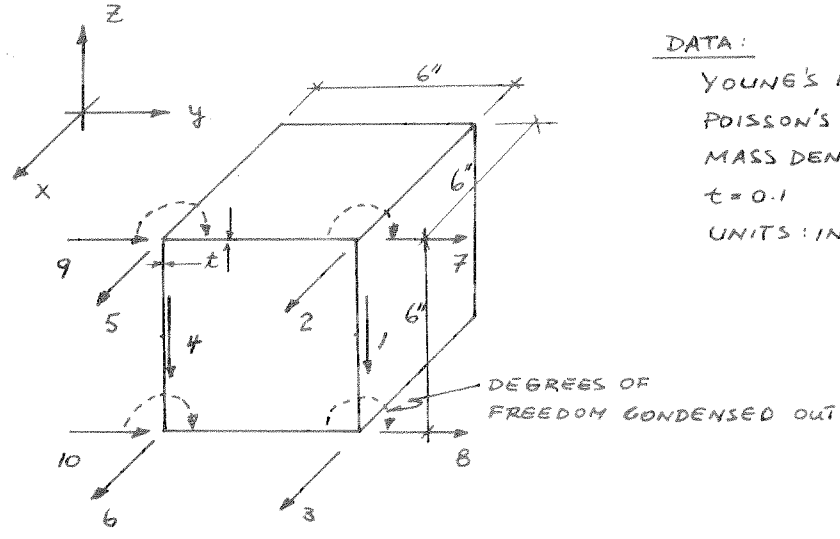
Table 3 gives the value $t_i^{(k+1)}$ in Eq. (4.17) calculated by program SSPACE. We note that after 6 iterations we had convergence. Also, as was found when INTVAL was used, we had approximations to all 8 smallest eigenvalues.

Program INTVAL is a version of SSPACE to calculate all eigenvalues in an interval (see Section 4.7). The program was used to find the eigenvalues between 0.0 and 10.0 and then between 10.0 and 40.0. In both analyses the program performed seven times the iterations a, b and then used the iteration c, in which eigenvalue approximations are obtained. In order to find the lowest 4 eigenvalues, two iterations c were performed. In the calculation of the higher eigenvalues individual vector iterations were carried out on two vectors after the second iteration c and then convergence was reached in one more iteration c.

In this analysis SECANTD is faster than BANEIG II and as fast



(a) ELEVATION OF CANTILEVER BOX



DATA:
YOUNG'S MODULUS = 30 000
POISSON'S RATIO = 0.25
MASS DENSITY = 1/3.6
$t = 0.1$
UNITS: IN, KIP

(b) TYPICAL SEGMENT BETWEEN SECTIONS A AND B WITH DEGREES OF FREEDOM AT SECTION A

FIGURE 10: FINITE ELEMENT IDEALIZATION OF CANTILEVER BOX

TABLE 2: CP times used in the analysis of the cantilever box

$n = 100$; $(m_A)_{max} = 19$; $m_B = 0$

PROGRAM	TIME [sec]								
	λ_1=0.5503..	λ_2=0.5754..	λ_3=1.803..	λ_4=6.783..	λ_5=11.11..	λ_6=15.46..	λ_7=17.43..	λ_8=37.14..	TOTAL
SECANTD	2.48	0.66	0.82	1.43	1.27				6.66
BANEIG II	3.33	1.83	1.81	2.71	2.43				12.46
SSPACE	calculated together								6.40
INTVAL	calculated together								6.00
INTVAL					calculated together				7.44

TABLE 3: Convergence characteristics of subspace iteration analyses

ANALYSIS	NO OF ITERATION	$t_i^{(k+1)} = \lvert\lambda_i^{(k+1)} - \lambda_i^{(k)}\rvert / \lambda_i^{(k+1)}$							
		i = 1	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7	i = 8
CANTILEVER BOX	6	0.6E-14	0.5E-13	0.4E-14	0.5E-09	0.3E-08	0.5E-06	0.2E-05	0.5E-03
	7	0.0	0.0	0.2E-13	0.2E-11	0.4E-10	0.2E-07	0.1E-06	0.7E-04
STURM LIOUVILLE PROBLEM	6	0.2E-11	0.7E-06	0.2E-04	0.3E-02	0.7E-03	0.5E-01	0.8E-01	0.2E+00
	7	0.2E-13	0.2E-07	0.2E-05	0.7E-03	0.1E-03	0.2E-01	0.3E-01	0.1E+00
	8	0.2E-13	0.7E-09	0.2E-06	0.1E-03	0.2E-04	0.1E-01	0.1E-01	0.7E-01
	9	0.2E-13	0.2E-10	0.1E-07	0.2E-04	0.4E-05	0.5E-02	0.6E-02	0.4E-01
	10	0.6E-14	0.7E-12	0.1E-08	0.4E-05	0.8E-06	0.2E-02	0.3E-02	0.2E-01
	11	0.6E-14	0.3E-13	0.1E-09	0.7E-06	0.2E-06	0.1E-02	0.2E-02	0.2E-01
DAM	6	0.7E-05	0.1E-01	0.6E-03	0.1E-03	0.9E-03	0.3E-01		
	7	0.4E-06	0.1E-02	0.7E-04	0.4E-04	0.3E-03	0.9E-02		
	8	0.2E-07	0.1E-03	0.9E-05	0.1E-04	0.8E-04	0.3E-02		
	9	0.1E-08	0.1E-04	0.1E-05	0.5E-05	0.3E-04	0.1E-02		
	10	0.1E-09	0.2E-05	0.2E-06	0.2E-05	0.1E-04	0.4E-03		
	11	0.1E-10	0.2E-06	0.2E-07	0.7E-06	0.6E-05	0.1E-03		
FRAME	3	0.8E-06	0.5E-03	0.8E-02	0.8E-01	0.9E-01	0.1E+00		
	4	0.5E-10	0.2E-05	0.3E-03	0.8E-02	0.3E-01	0.4E-01		
	5	0.	0.1E-07	0.1E-04	0.1E-02	0.9E-02	0.2E-01		
	6	0.2E-13	0.7E-10	0.7E-06	0.3E-03	0.3E-02	0.7E-02		
BUILDING	3	0.6E-02	0.6E-02	0.2E-01	0.4E-01	0.1E+00	0.4E+00	0.9E+00	0.5E+02
	4	0.6E-04	0.7E-04	0.4E-03	0.3E-02	0.7E-02	0.1E+00	0.4E-01	0.6E-01
	5	0.7E-06	0.1E-05	0.1E-04	0.2E-03	0.7E-03	0.2E-01	0.9E-02	0.9E-02
	6	0.8E-08	0.3E-07	0.4E-06	0.2E-04	0.1E-03	0.5E-02	0.2E-02	0.5E-02
	7	0.1E-09	0.8E-09	0.1E-07	0.2E-05	0.2E-04	0.1E-02	0.6E-03	0.4E-02
	8	0.1E-11	0.2E-10	0.6E-09	0.2E-06	0.5E-05	0.5E-03	0.2E-03	0.3E-02

as the subspace iteration analysis. We also note that the cut-off analysis for all eigenvalues below 10.0 needs about the same time as the analysis with SSPACE.

6.3 Sturm-Liouville Problem

In this example matrices A and B were obtained in a finite element solution of the Sturm-Liouville problem. The order of the matrices was 200 and the half bandwidth in both was 3. Table 4 gives the smallest 4 eigenvalues and the iteration times used by programs SECANT and SSPACE. The convergence in the subspace iteration is shown in Table 3. The program iterated with 8 vectors and we obtained approximations to the smallest 8 eigenvalues.

As expected, at this small bandwidth SECANT is faster than SSPACE. Note that BANEIG II could not be used.

6.4 Analysis of Dam

Figure 11 shows the finite element idealization of the dam with the three-dimensional element used. Only half of the dam was considered to calculate the symmetric eigenmodes. The order of the system was 213 with maximum half bandwidth 59. A lumped mass formulation was used with masses at all degrees of freedom.

Table 5 gives the three smallest eigenvalues of the system with the iteration times taken by programs SECANTD, BANEIG II and SSPACE. In this example, because of the larger bandwidth, SSPACE is more efficient, but not excessively. An older program was once used to analyze the dam for the lowest 6 eigenvalues. From

TABLE 4: CP time used in the analysis of the Sturm Liouville problem

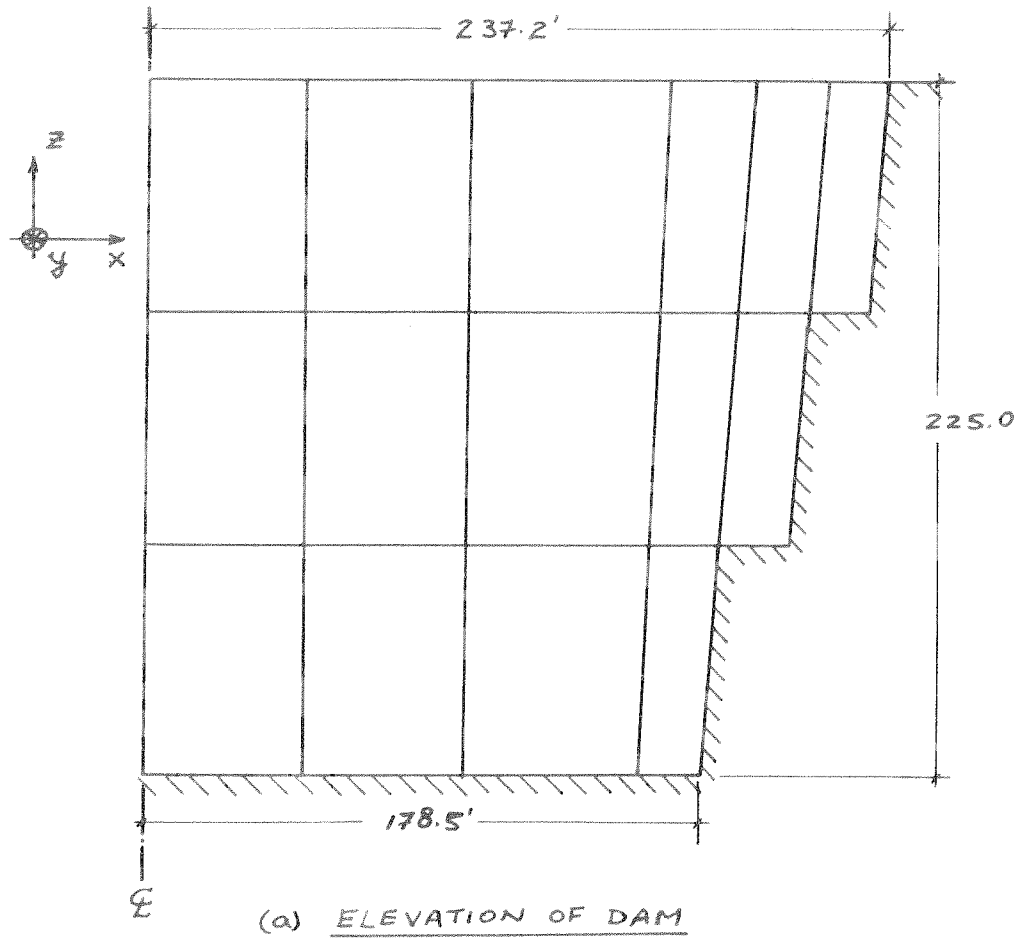
$n = 200,\ m_A = m_B = 3$

PROGRAM	TIME [sec]				
	$\lambda_1 = 79.7492..$	$\lambda_2 = 238.742..$	$\lambda_3 = 396.722..$	$\lambda_4 = 553.677$	TOTAL
SECANT	1.81	1.31	1.32	1.35	5.79
SSPACE	calculated together				16.69

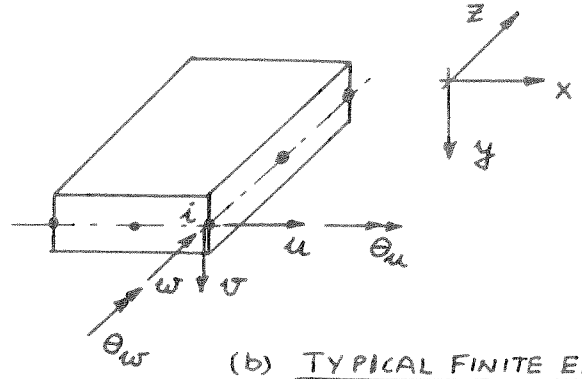
TABLE 5: CP time used in the analysis of the dam

$n = 213,\ (m_A)_{max} = 59,\ m_B = 0$

PROGRAM	TIME [sec]			
	$\lambda_1 = 2.18965..$	$\lambda_2 = 3.35626..$	$\lambda_3 = 3.72271..$	TOTAL
SECANTD	34.32	24.83	12.08	71.23
BANEIG II	39.25	29.69	22.00	91.82
SSPACE	calculated together			38.33



(a) ELEVATION OF DAM



(b) TYPICAL FINITE ELEMENT AND DEGREES OF FREEDOM AT NODE i

DATA:

YOUNG'S MODULUS = 432000

MASS DENSITY = 1.0

POISSON'S RATIO = 0.20

UNITS : FT, KIP

FIGURE 11: FINITE ELEMENT IDEALIZATION OF DAM

those results we know that at convergence in this analysis we had actually approximations to the lowest 6 eigenvalues.

6.5 Analysis of Plane Frame

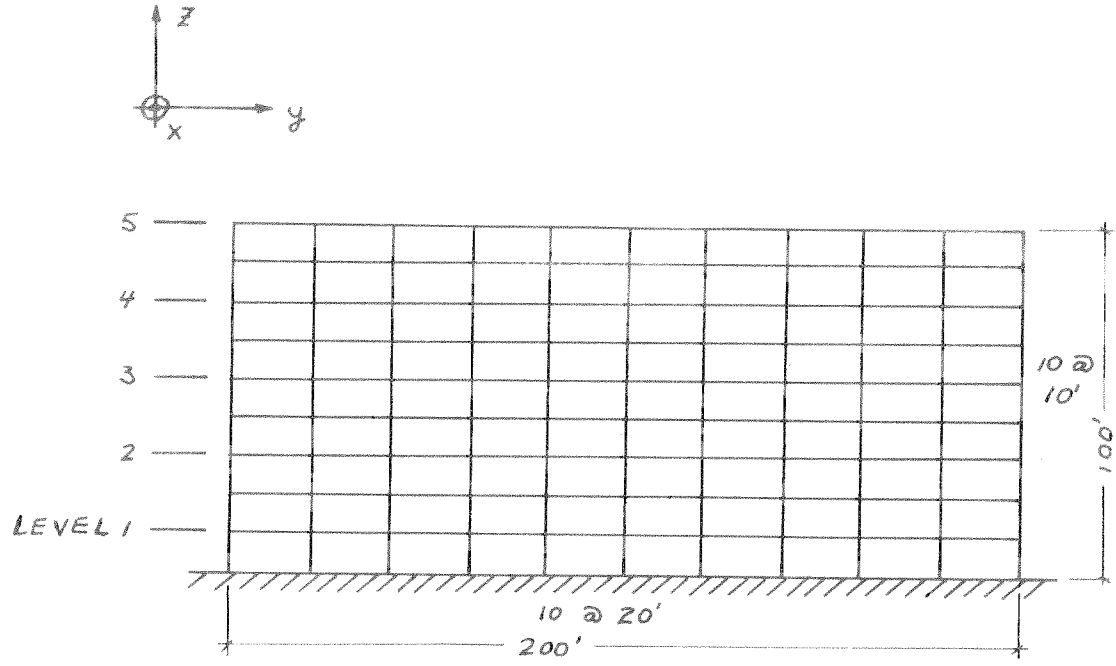
A 9-story high and 10-bay long plane frame was analyzed. The details of the frame are given in Fig. 12. The stiffness matrix, of order 297 and maximum half bandwidth 29, and the lumped mass matrix were generated by program SAP [19]. We note that zero masses were associated with the rotational degrees of freedom and BANEIG II could not be included in the comparison.

To calculate the 3 smallest eigenvalues program SECANTD performed an in-core solution and SSPACEB used 3 blocks in a subspace iteration. The eigenvalues with the iteration times are given in Table 6. Referring to Table 3 we note the relatively fast convergence of the subspace iteration.

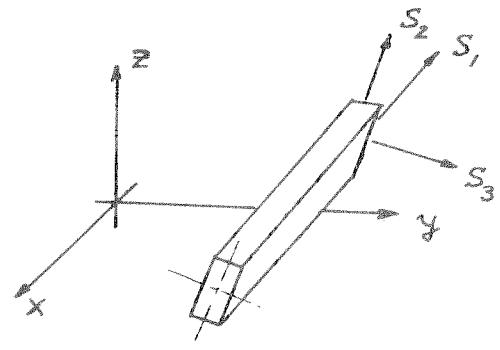
A Ritz analysis with five transformation vectors obtained by applying unit loads into the y-direction at levels 1,2,3,4 and 5 gave $\lambda_1 = 0.6113$, $\lambda_2 = 7.320$, $\lambda_3 = 30.08$. We should note that the projected operator B (the generalized mass matrix) is quite ill-conditioned with respect to inversion, because the unit loads give much the same Ritz transformation vectors. This ill-conditioning is also present in the first subspace iteration in SSPACEB.

6.6 Analysis of Three-Dimensional Building Frame

The complex building frame shown in Fig. 13 was analyzed using program SSPACEB. At each joint beams are spanning into the



(a) ELEVATION OF FRAME



DATA:

YOUNG'S MODULUS = 432000
MASS DENSITY = 1.0
FOR ALL BEAMS AND COLUMNS $A_1 = 3.0$,
$I_1 = I_2 = I_3 = 1.0$
UNITS: FT, KIP

(b) BEAM LOCAL AXES S_1, S_2, S_3

A_1 = AREA ASSOCIATED WITH S_1
I_1, I_2, I_3 = FLEXURAL INERTIA ABOUT S_1, S_2, S_3 RESP.

FIGURE 12: LAYOUT OF 9-STORY 10-BAY PLANE FRAME

TABLE 6: CP time used in the analysis of the plane frame

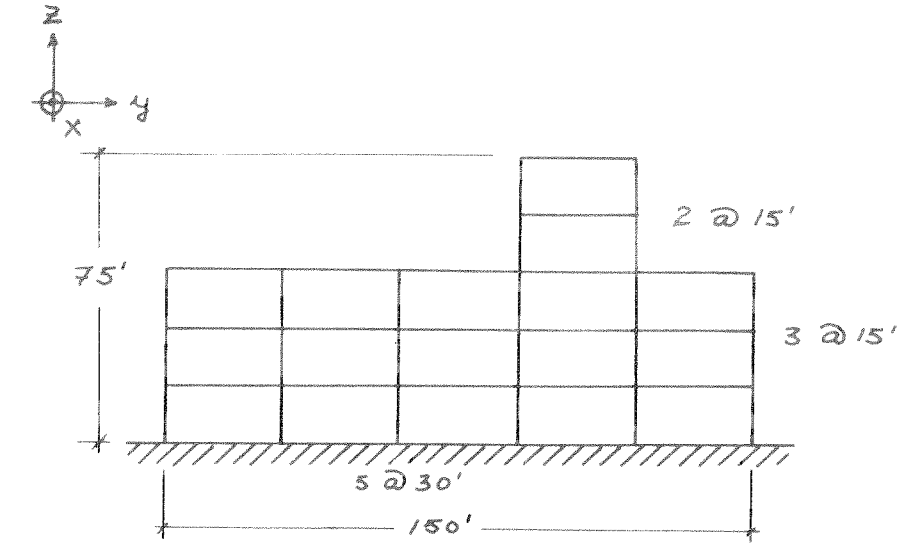
$n = 297$, $(m_A)_{max} = 29$, $m_B = 0$

PROGRAM	TIME [sec]			
	$\lambda_1 = 0.589541..$	$\lambda_2 = 5.52695..$	$\lambda_3 = 16.5878..$	TOTAL
SECANTD	9.06	12.30	18.24	39.60
SSPACEB	calculated together			24.48

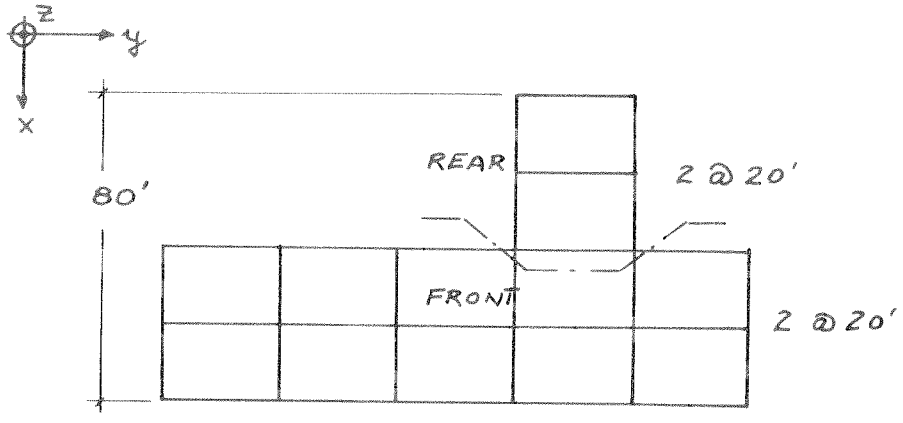
TABLE 7: CP time used in the analysis of the building frame

$n = 468$, $(m_A)_{max} = 155$, $m_B = 0$

PROGRAM	TIME [sec]				
	$\lambda_1 = 0.41537..$	$\lambda_2 = 0.54930..$	$\lambda_3 = 0.78606..$	$\lambda_4 = 1.0325..$	TOTAL
SSPACEB	calculated together				159.59



(a) ELEVATION OF BUILDING



(b) PLAN OF BUILDING

DATA : (FOR NOTATION REFER TO FIG. 12)

YOUNG'S MODULUS = 432000 , MASS DENSITY = 1.0

COLUMNS IN FRONT BUILDING $A_1 = 3.0$, $I_1 = I_2 = I_3 = 1.0$

COLUMNS IN REAR BUILDING $A_1 = 4.0$, $I_1 = I_2 = I_3 = 1.25$

ALL BEAMS INTO X-DIRECTION $A_1 = 2.0$, $I_1 = I_2 = I_3 = 0.75$

ALL BEAMS INTO Y-DIRECTION $A_1 = 3.0$, $I_1 = I_2 = I_3 = 1.0$

UNITS : FT, KIP

FIGURE 13: LAYOUT OF THREE DIMENSIONAL BUILDING FRAME

x, y and z directions and we have 6 degrees of freedom. Only beam elements have been used in the finite element idealization. Note that the column stiffnesses are different in the front and the rear part of the frame. Program SAP assembled the structure stiffness and lumped mass matrix. There were no masses associated with the rotational degrees of freedom. The order of the stiffness matrix was 468 and the half bandwidth varied from 59 to 155. In the analysis 13 blocks with 37 degrees of freedom per block have been used.

The smallest four eigenvalues and the solution time used are given in Table 7. A factorization took about twice as much time as a subspace iteration. Table 8 gives the eigenvalues calculated in each subspace iteration. Already in the third iteration we have good approximations to the eigenvalues sought. Table 3 shows the convergence in the iteration.

6.7 Conclusions from Example Analyses

The analyses confirm that the determinant search technique is more economical than the subspace iteration on systems with small bandwidth. Further, on those examples which BANEIG II could also analyze, the determinant search technique was as efficient.

In the subspace iterations we always obtained monotonic convergence to the required eigenvalues and associated vectors. The starting subspace was described in Section 4.4 and Eq. (4.16) was used for the dimension of the subspace. About 8 iterations were needed for convergence to RTOL = 10^{-6} in Eq. (4.17).

TABLE 8: Eigenvalue approximations calculated in the analysis of the building

NO OF ITERATION	EIGENVALUE APPROXIMATIONS							
1	0.5206	0.9007	1.329	1.869	4.320	7.550	23.67	296.7
2	0.4117	0.5529	0.7992	1.075	1.676	3.002	4.666	194.9
3	0.4154	0.5493	0.7864	1.035	1.498	2.210	2.395	3.656
4	"	"	0.7861	1.033	1.488	2.008	2.293	3.463
5	"	"	"	"	1.487	1.971	2.272	3.432
6	"	"	"	"	"	1.962	2.268	3.415
7	"	"	"	"	"	1.959	2.266	3.403
8	"	"	"	"	"	1.958	"	3.391

Note that, as in Table 8, the eigenvalue approximations calculated in each iteration are upper bounds to the 'exact' eigenvalues of the system (see Section 4.2).

In Table 9 time estimates are given to solve for p eigenvalues and corresponding vectors. If we put p equal to the number of eigenvalues solved for in the analyses, we obtain the times reported in the previous tables. Of the determinant search technique and the subspace iteration analysis only the one which is preferably used is included. The estimates for an in-core Householder-QR-Inverse iteration solution given in the table have been obtained by extrapolating the values in Table 1 of Reference 1, using that it is an n^3 process. The largest problem solved in [1] on the CDC 6400 was n = 100. Using the method it is assumed in Table 9 that we solve for all eigenvalues and only the required eigenvectors. Naturally, all these time estimates can only be approximate.

At the end of Section 2.2 it was described that we want to consider an eigenvalue problem as large if it is much cheaper to solve for only the required eigenvalues than to calculate simply all. Table 9 shows that the cantilever box and the dam example are in this sense not large problems. However, the other three problems are certainly large. Note that the plane and building frame could not be analyzed using BANEIG or the QR method. Also, because of the very small bandwidth, the determinant search solution of the Sturm-Liouville problem is estimated to be more economical than the Householder-QR-Inverse iteration solution even if all eigenvalues and vectors are required.

TABLE 9: Estimated times in sec for solution of p eigenvalues and corresponding vectors in example analyses

EXAMPLE	METHOD			
	BANEIG	QR	DETERMINANT SEARCH**	SUBSPACE ITERN**
CANTILEVER BOX	2.5 p	18.3+0.28 p	1.3 p	
STURM-LIOUVILLE PROBLEM	NOT POSSIBLE	146.0+2.2 p*	1.5 p	
DAM	30.6 p	177.0+2.7 p	23.7 p	
PLANE FRAME	NOT POSSIBLE	NOT POSSIBLE	13.2 p	
BUILDG. FRAME	NOT POSSIBLE	NOT POSSIBLE		53.7+26.5 p

*time taken for transformation to standard eigenvalue problem has been neglected.

**only the algorithm which is preferably used is included in the comparison.

Consider the example of the building frame to compare with the information given in Fig. 8. In this case Table 9 shows that the subspace iterations for one eigenvalue are in operations about equivalent to one factorization. In Fig. 8 we would need to use as an equivalent full half bandwidth about 80 to obtain the same operation estimate. The actual half bandwidth of the system varied between 59 and 155 with many zeros within the band, so that the estimate of 80 is reasonable.

We should also note that to calculate in the determinant search solutions an eigenpair (λ_i, v_i), the number of required factorizations varied between 1 and 8 and the number of inverse iterations varied between 2 and 12. The average number of factorizations and inverse iterations was five.

7. SUMMARY AND CONCLUSIONS

Programs have been developed for the solution of a few eigenvalues and associated vectors in the generalized eigenvalue problem $Av = \lambda Bv$, when the order of the matrices is large. We were concerned with the problem as it arises in structural analysis when matrix A is positive definite. Two different techniques have been implemented, a determinant search algorithm and a subspace iteration which both solve the generalized eigenvalue problem directly without a transformation to the standard form.

In the past determinant search techniques have generally been considered inefficient because many factorizations are needed. In this research an algorithm has been developed which on practical problems in terms of speed is as efficient as latest routines available. However, the advantages are that B can be diagonal non-negative definite or banded. Also, high accuracy in the eigenvectors is not required for numerical stability of the iteration process. In dynamic analysis, the case B diagonal with zero or small elements is very important, but no direct efficient solution routine was available. If in the determinant search solution B is banded little extra work is required. Therefore, the eigenvalue problem in consistant mass formulation is not much more expensive than in a lumped mass analysis.

The determinant search technique is used best as an in-core solver on systems with small to medium bandwidth. For systems

with large bandwidth an efficient subspace iteration algorithm has been presented. This algorithm was to my knowledge not implemented before. The iteration is carried out on both operators A and B simultaneously. The starting subspace is chosen automatically from the elements in A and B. The small generalized eigenvalue problem obtained when the projections of A and B are calculated is solved using a generalized Jacobi iteration method. This method iterates on both operators simultaneously taking advantage of the fact that they tend towards diagonal form in the subspace iterations. Also, the projection of B may be ill-conditioned with respect to inversion without introducing numerical difficulties. After convergence of the subspace iteration a Sturm Sequence check can be performed to assure that the required eigenvalues have been found. The algorithm was used to find the smallest eigenvalues and corresponding vectors or to calculate all eigenvalues in a specified interval. As in the determinant search solution, matrix B may be banded or diagonal non-negative definite. Also, eigenvalues need not be found to high precision.

In order to estimate the cost of the eigenvalue solution the number of factorizations equivalent to the subspace iterations have been presented. In example analyses about 8 iterations were required for convergence to about 5 digit accuracy. In terms of operations this means that with B diagonal the subspace iterations required to find the 5 smallest eigenvalues and corresponding vectors are equivalent to about one factorization

when the half bandwidth of A is 320.

The convergence characteristics of the algorithms have been observed in example analyses. In the subspace iterations we always obtained monotonic convergence to the required eigenvectors. In the analyses we used the programs in Appendix II which solve for the smallest eigenvalues and corresponding vectors. This is the important problem arising in earthquake analysis. As shown in the thesis, with small modifications, the algorithms may also be used to find the largest eigenvalues in the problem $Bv = \mu Av$ arising in bifurcation buckling.

Using the conventional Ritz analysis, which includes the static condensation analysis, to obtain the smallest eigenvalues and associated vectors we have no idea of how accurate the solution approximates the 'exact' required eigensystem. In fact, it may well be that an important eigenvalue and corresponding vector is not approximated at all. The main aim in this research was to provide programs with which we can solve efficiently to the required precision for the eigenvalues and corresponding vectors, and to give cost estimates. Naturally, the cost to solve accurately for the eigensystem must be higher than a very approximate analysis.

For practical application the important program is MODES which solves for the smallest eigenvalues and associated vectors when B is diagonal non-negative definite and matrix A has any size and bandwidth. The program could be modified to solve the generalized eigenvalue problem when B is banded and for the

solution of buckling problems.

In this dissertation the eigenvalue problem $Av = \lambda Bv$ was considered when both matrices are symmetric and at least A is positive definite. The important and very frequent cases occurring in practice in which this eigenvalue problem needs to be solved have been discussed in Chapter 2. In conclusion we should mention the quadratic eigenvalue problem

$$(\lambda^2 A_2 + \lambda A_1 + A_o)\, x = 0 \tag{7.1}$$

which can also be reduced to the form $Av = \lambda Bv$. In Eq. (7.1) A_2, A_1 and A_o are symmetric and most commonly positive definite matrices of order n. This eigenvalue problem arises in dynamic analysis of structures when non-proportional damping is present, in which case C* in Eq. (2.5) is not diagonal [20].

To identify the eigenvalue problem as a problem of form $Av = \lambda Bv$, we rewrite Eq. (7.1) as

$$\begin{bmatrix} 0 & A_o \\ A_o & A_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} A_o & 0 \\ 0 & -A_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{7.2}$$

where we observe that A and B are not positive definite. The solution routines developed in this research are therefore not applicable.

The solution of the quadratic eigenvalue problem yields 2n complex eigenvalues and corresponding eigenvectors. Efficient algorithms for this problem are available when the order of the matrices is small [5]. Muller's method of successive quadratic

interpolation can be applied in a determinant search technique to find the zeroes of the function $p(\lambda) = \det(\lambda^2 A_o + \lambda A_1 + A_o)$. When the complete eigensystem is wanted, Eq. (7.2) is conveniently written in the form of the standard eigenvalue problem

$$\begin{bmatrix} 0 & I \\ -A_2^{-1}A_o & -A_2^{-1}A_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix} \tag{7.3}$$

This has the disadvantage of working with a matrix of order 2n, but we can now use the efficient QR algorithm for nonsymmetric matrices.

Currently, in practice the order of the eigenvalue problem is usually small,but larger systems may need to be solved. In this case only some eigenvalues and corresponding vectors may be required. Then in future research we could extend this work and develop efficient solution routines which calculate only the few required eigenvalues and corresponding vectors in Eq. (7.1), when the order of the matrices is large. These algorithms should also give in this case a much more economical solution than to solve simply for the complete eigensystem.

REFERENCES

1. Clough, R. W., "Analysis of Structural Vibrations and Dynamic Response," Proceedings of U.S.-Japan Symposium on Recent Advances in Matrix Methods of Structural Analysis and Design, Tokyo, Japan, 1969.

2. Wilson, E. L., "Earthquake Analysis of Reactor Structures," Proceedings Symposium on Seismic Analysis of Pressure Vessels and Piping Components, The American Society of Mech. Eng., 1971.

3. Uhrig, R., "Reduction of the Number of Unknowns in the Displacement Method Applied to Kinetic Problems," J. Sound and Vib. (1966), 4(2).

4. Rubinstein, M. F., Structural Systems-Statics, Dynamics and Stability, Prentice-Hall, 1970.

5. Wilkinson, J. H., The Algebraic Eigenvalue Problem, Clarendon Press, Oxford.

6. Ostrowski, A. M., "On the Convergence of the Rayleigh Quotient Iteration for the Computation of the Characteristic Roots and Vectors," Parts I-VI, Arch. Rat. Mech. & Anal., 1-4.

7. Felippa, C. A., "BANEIG-Eigenvalue Routine for Symmetric Band Matrices," Computer Programming Series, Division of Structural Engineering and Structural Mechanics, University of California, Berkeley, July 1966.

8. Rutishauser, H., "Deflation bei Bandmatrizen," Zamp, Vol. 10, 1959.

9. Courant, R., and Hilbert, D., Methods of Mathematical Physics, Vol. 1, Interscience Publishers, New York, 1953.

10. Gupta, K. K., "Vibration of Frames and Other Structures with Banded Stiffness Matrix," Int. Jour. Num. Methods, Vol. 2, 1970.

11. Schwarz, H. R., Numerik Symmetrischer Matrizen, B. G. Teubner, Stuttgart.

12. Rutishauser, H., "Computational Aspects of F. L. Bauer's Simultaneous Iteration Method," Numer. Math. 13, 1969.

13. Collatz, L., The Numerical Treatment of Differential Equations, Springer, New York.

14. Bauer, F. L., "Das Verfahren der Treppen-Iteration und verwandte Verfahren zur Lösung Algebraischer Eigenwertprobleme, Zamp 8, 1957.

15. Jennings, A., "A Direct Iteration Method of Obtaining Latent Roots and Vectors of a Symmetric Matrix," Proc. Camb. Phil. Soc., Vol. 63, 1963.

16. Brönlund, O. E., "Eigenvalues of Large Matrices," Symp. on Finite Element Techniques at the Institut für Statik and Dynamik der Luft und Raumfahrtskonstruktionen, University of Stuttgart, June 1969

17. Jennings, A., and Orr, R. L., "Application of the Simultaneous Iteration Method to Undamped Vibration Problems," Int. Journ. for Num. Methods in Eng., Vol. 3, 1971.

18. Falk, S., and Langemeyer, P., "Das Jacobische Rotationsverfahren für reell-symmetrische Matrizenpaare," Elektron. Datenverarb., 1960.

19. Wilson, E. L., "SAP - A General Structural Analysis Program," Structural Engineering Laboratory Report No. 70-20, University of California, Berkeley, September 1970

20. Hurty, W., and Rubinstein, M. F., Dynamics of Structures, Prentice-Hall, 1964.

21. Cheung, Y. K., "Folded Plate Structures by Finite Strip Method," Journ. of Struct. Div., ASCE, Vol. 95, Dec. 1969

22. Parlett, B. N., and Kahan, W., "On the Convergence of a Practical QR Algorithm," Proceedings of the IFIP Congress, 1968.

APPENDIX I

SOLUTION OF SMALL SYMMETRIC EIGENVALUE PROBLEMS

Solution techniques for small eigenvalue problems of the form $Av = \lambda v$ have now reached a relatively high standard of efficiency. Although there are many different methods available, we only need to choose between a few procedures for the practical solution of an eigenvalue problem. The aim in this script is to present briefly these methods of solution and at the same time also recall to the reader some background theory. Naturally, no complete treatment is possible. For reference see [5] [11].

I.1 Facts from Linear Algebra

We consider the eigenvalue problem

$$Av = \lambda v \tag{1}$$

where A is a symmetric real matrix of order n. It is the representation of a linear transformation in an n-dimensional vector space after having chosen a basis. There are n scalars for λ with corresponding vectors which satisfy Eq. (1). The scalars are eigenvalues and are the zeros of the polynomial $p(\lambda) = \det (A - \lambda I)$, where

$$|\lambda_1| \leq |\lambda_2| \leq |\lambda_3| \leq \ldots \leq |\lambda_n|$$

It can be shown that all eigenvalues are real and that there are n orthonormal eigenvectors v_i, meaning $v_i^T v_j = \delta_{ij}$

(= Kronecker delta). The eigenvectors are bases of the eigenspaces corresponding to the eigenvalues. The dimension of an eigenspace is equal to the multiplicity of the corresponding eigenvalue. The bases in multiple dimension eigenspaces are not unique. We note that for an eigenspace the action of the operator A is simply scalar multiplication.

The n solutions to Eq. (1) may be written as

$$AV = V\Lambda \tag{2}$$

where $\Lambda = \text{diag}\ (\lambda_i)$ and the columns of V are the eigenvectors.

Using the orthonormality of the eigenvectors we obtain

$$V^T AV = \Lambda \tag{3}$$

and

$$A = V\Lambda V^T \tag{4}$$

In Eq. (3) we perform a change of basis using an orthogonal similarity transformation. Λ represents the same linear transformation as A but in the basis of eigenvectors. Equation (4) shows that A is completely defined once we know the eigenvalues and corresponding eigenvectors.

As an example consider the discretization of a continuum by 'finite elements'. The resulting n equilibrium equations are

$$R = Ku$$

in which K is a linear transformation which transforms generalized displacements into generalized forces. The basis in the n-dimensional space is formed by the n element displacement functions, which are associated with n generalized force and

displacement coordinates. These are listed in vectors R and u, respectively. Each point in the space represents a displacement configuration of the continuum. If a different basis is chosen, the same linear transformation is represented by a different matrix. In particular, if we can choose the eigenvectors as a basis, then K is diagonal and the solution of the equilibrium equations is trivial. In general, the eigenvectors are not known a priori, but the idea is used in the Finite Strip Method [21].

In practice, requirements for eigenvalues and vectors vary. We may want a few or all eigenvalues and similarly only a few or all eigenvectors.

I.2 Solution Techniques

We note that any algorithm for calculating eigenvalues is an infinite process. This follows because, in general, the roots of polynomials of degree greater than four cannot be calculated without iteration. Basically, there are three different groupings for methods of solution.

I.2.1 Polynomial Iterations

A polynomial root finder could be used once the coefficients of $p(\mu)$ have been solved for. This technique has proved unstable, because small errors in the coefficients of the polynomial give large errors in the eigenvalues. On the other hand, we can use the polynomial implicitely as in a determinant search technique and obtain stable and reliable procedures (see Chapter 3).

I.2.2 Vector Iterations

In the power method we find an eigenvalue and the associated vector. Let x_1 be an arbitrary vector, then the iteration is defined by

$$x_{k+1} = Ax_k \qquad k = 1,2,3,\ldots \tag{5}$$
$$= A^k x_1$$

In practice x_{k+1} is normalized to have unit length. This does not affect a convergence study, as an eigenvector is only defined by its direction. Let

$$x_1 = \sum_{i=1}^{n} \alpha_i v_i \quad ; \quad \alpha_i \neq 0 \text{ all } i$$

then

$$A^k x_1 = \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i$$

$$= \lambda_n^k \left\{ \alpha_n v_n + \sum_{i=1}^{n-1} \alpha_i \left(\frac{\lambda_i}{\lambda_n}\right)^k v_i \right\}$$

$$\rightarrow \lambda_n^k \alpha_n v_n \quad \text{as} \quad k \rightarrow \infty$$

The iteration converges ultimately with a rate $\left|\frac{\lambda_{n-1}}{\lambda_n}\right|$ towards the eigenvector corresponding to λ_n. When λ_{n-1} is nearly equal to λ_n we get slow convergence. In case $\lambda_{n-1} = \lambda_n$ the iteration will give a vector in the subspace spanned by v_n and v_{n-1}. But when $|\lambda_{n-1}| = |\lambda_n|$ and $\lambda_{n-1} \neq \lambda_n$ we do not converge to an eigenvector.

To obtain an approximation to λ_n once we have an approximate eigenvector x_k, we enter Eq. (5) with $||x_k||_2 = 1.0$ and calculate

$|\lambda_n| \doteq ||x_{k+1}||_2$.

Closely related to the power method is the inverse iteration, defined by

$$Ax_{k+1} = x_k \qquad k = 1,2,... \tag{6}$$

or

$$x_{k+1} = A^{-k} x_1$$

The dominance of the eigenvalues is now reversed and convergence is linear with the rate $\left|\frac{\lambda_1}{\lambda_2}\right|$ towards the eigenvector corresponding to λ_1.

In numerical calculations we can speed up convergence by suitable origin shifts. We note that the operator $(A - \mu I)$ has the eigenvalues $\lambda_1 - \mu$, $\lambda_2 - \mu, \ldots, \lambda_n - \mu$. Let

$$|\lambda_p - \mu| = \max_i |\lambda_i - \mu|$$

and

$$|\lambda_q - \mu| = \min_i |\lambda_i - \mu|$$

then for the shifted operator the power method and the inverse iteration converge with the ratios

$$\frac{\max_{i \neq p} |\lambda_i - \mu|}{|\lambda_p - \mu|} \quad \text{and} \quad \frac{|\lambda_q - \mu|}{\min_{i \neq q} |\lambda_i - \mu|}$$

respectively. In inverse iteration as μ gets close to λ_q we obtain better convergence. But in the power method, the convergence rate is bounded. For instance, assume that $\lambda_i > 0$ for all i then for best convergence towards the eigenvector corresponding to λ_n, we use $\mu = \frac{\lambda_1 + \lambda_{n-1}}{2}$.

The convergence rate in the inverse iteration can be much increased by choosing the Rayleigh quotient as the shift. The Rayleigh quotient of a nonzero vector v is defined as

$$\rho(v) = \frac{(v,Av)}{(v,v)} \tag{7}$$

Geometrically, the distance between Av and μv expressed as $||Av - \mu v||_2$ is minimized when $\mu = \rho(v)$.

If v is a first order approximation to an eigenvector, then $\rho(v)$ is a second order approximation to the corresponding eigenvalue. Namely, assume $v = \sum_{i=1}^{n} \alpha_i v_i$, where $|\frac{\alpha_i}{\alpha_r}| \ll 1 \ (i \neq r)$ and use $\epsilon_i = \frac{\alpha_i}{\alpha_r}$, $\sum = \sum_{\substack{i=1 \\ i \neq r}}^{n}$, then

$$\rho(v) = (\lambda_r + \sum \epsilon_i^2 \lambda_i)/(1 + \sum \epsilon_i^2)$$

$$\doteq (\lambda_r + \sum \epsilon_i^2 \lambda_i)(1 - \sum \epsilon_i^2 + 0(4))$$

$$\doteq (\lambda_r + \sum \epsilon_i^2 \lambda_i)$$

In the inverse iteration different strategies can be followed. We may apply a Rayleigh quotient shift only a few times or we may shift in each iteration. In this case we have the Rayleigh quotient iteration, defined as

$$(A - \rho(x_k)I)\, x_{k+1} = x_k \ \ell_k \qquad (8)$$

where ℓ_k is chosen to have $||x_{k+1}||_2 = 1$. Under conditions $(\rho(x_k), x_k)$ converges in the limit cubically to an eigenpair [22]. Essentially this follows because in Eq. (6) the eigenvector converges linearly but is here used in each iteration to obtain a second order approximation to the eigenvalue and shift.

Assume that an eigenvalue λ_i and its vector v_i have been calculated. For iteration towards another eigenvalue, we need to deflate either the matrix or the iteration vectors.

A stable matrix deflation can be carried out by finding an orthogonal matrix P whose first column is the calculated eigenvector. Then

$$P^T A P = \begin{bmatrix} \lambda_i & 0 \\ 0 & A_1 \end{bmatrix}$$

and A_1 has all the remaining eigenvalues of A.

A vector deflation is simply carried out by orthogonalizing the iteration vector to the calculated eigenvector. Using the Gram-Schmidt process we have

$$x_{k+1} = \bar{x}_{k+1} - \alpha_i v_i$$

$$\alpha_i = (\bar{x}_{k+1}, v_i)$$

and $\bar{x}_{k+1}$ not x_{k+1} is obtained in the algorithms of Eqs. (5), (6), and (8). In the Rayleigh quotient iteration it can be sufficient to orthogonalize only x_1.

The vector iteration techniques above are only economical when a few eigenvalues and vectors are required. But inverse iteration is used efficiently to find eigenvectors when the eigenvalues have been calculated by some other procedure, for example by the QR iteration.

I.2.3 Transformation Methods

Eq. (3) suggests to find transformation matrices which by successive application transform A into diagonal form. In this

iteration each transformation must be a similarity transformation

$$A_{k+1} = W^{-1} A_k W$$

where A_{k+1} has the same eigenvalues as A_k. Particularly easy to use and stable are orthogonal similarity transformations, in which case $W^T W = I$.

A simple and very reliable transformation method is the Jacobi iteration. It gives at the same time the eigenvalues and an orthonormal set of eigenvectors. When the off-diagonal elements are small it is also efficient.

The iteration is defined as follows

$$A_{k+1} = P_k^T A_k P_k \qquad k = 1,2,\ldots \qquad A_1 = A \tag{9}$$

where

$$A_{k+1} \to \Lambda \quad \text{as} \quad k \to \infty$$

and the P_k are Jacobi rotation matrices,

$$P_k = \begin{bmatrix} 1 & & & & \\ & \cdot & & & \\ & & \cos\theta & -\sin\theta & \\ & & \sin\theta & \cos\theta & \\ & & & & \cdot \\ & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow \ell \\ \leftarrow m \\ \\ \end{matrix} \tag{10}$$

$$\qquad\qquad\quad \ell \qquad\quad m$$

The angle θ is selected to zero the element $a_{\ell m}^{(k)}$ in A_k

$$\tan 2\theta = \frac{2a_{\ell m}^{(k)}}{a_{\ell\ell}^{(k)} - a_{mm}^{(k)}}$$

The eigenvectors are simply the product of the orthogonal matrices used in the transformations. In practice, we need a

scheme for zeroing elements and we need a convergence criterion (see Section 4.3).

A very efficient and probably regarded as the best method for finding the eigensystem of a matrix is the Householder-QR Inverse iteration technique [22] [5].

The name hints at the three solution steps:

1. Householder transformations are used to reduce the matrix to tridiagonal form.

2. QR iteration yields the eigenvalues.

3. Using inverse iteration the eigenvectors of the tridiagonal matrix are calculated and transformed to the eigenvectors of A.

The reduction to tridiagonal form involves (n - 2) orthogonal similarity transformations

$$A_{k+1} = P_k^T A_k P_k \qquad k = 1,2,\ldots,n-2. \qquad A_1 = A \tag{11}$$

where

$$P_k = I - \theta w_k w_k^T \tag{12}$$

$$\theta = \frac{2}{w_k^T w_k}$$

We consider the case k = 1, which is typical. Let

$$P_1 = \begin{bmatrix} 1 & 0 \\ 0 & \bar{P}_1 \end{bmatrix} \quad ; \quad w_1 = \begin{bmatrix} 0 \\ \bar{w}_1 \end{bmatrix}$$

and

$$A_1 = \begin{bmatrix} a_{11} & a_1^T \\ a_1 & A_{11} \end{bmatrix}$$

where obviously A_{11} is a matrix of order $(n-1)$, $\bar{w}$, is a vector of order $(n-1)$, etc., then

$$P_1^T A_1 P = \left[\begin{array}{c|c} a_{11} & a_1^T \bar{P}_1 \\ \hline \bar{P}_1^T a_1 & \bar{P}_1^T A_{11} \bar{P}_1 \end{array}\right] \tag{13}$$

The vector $\bar{w}_1$ is determined from

$$(I - \theta \bar{w}_1 \bar{w}_1^T)\ a_1 = \pm \|a_1\|_2\ e_1 \tag{14}$$

where we can use wither + or - to avoid cancellation. Geometrically, we reflect a_1 in the hyperplane orthogonal to $\bar{w}_1$, where $\bar{w}_1$ is determined in such a way that only the first coordinate in the new vector in nonzero.

We only need to solve from Eq. (14) for a multiple of $\bar{w}_1$ and can use

$$\bar{w}_1 = a_1 + \text{sign}\ (a_{21})\ \|a_1\|_2\ e_1$$

Note that in Eq. (13) we do not need P_1 but only $\bar{w}_1$. The equivalent steps for $k = 2,\ldots,n-2$ are obvious.

Consider now the QR iteration on a general symmetric matrix A. Let

$$A = QR$$

where Q is an orthogonal and R an upper triangular matrix. This factorization could be obtained by applying the Gram-Schmidt process to the columns of A. We then form

$$RQ = Q^T AQ$$

and carried out an orthogonal similarity transformation on A. In the QR algorithm this process is repeated

$$A_k = Q_k R_k \tag{15}$$

$$A_{k+1} = R_k Q_k \qquad k = 1,2,\ldots \qquad A_1 = A \tag{16}$$

where

$$A_{k+1} \to \Lambda \quad \text{as } k \to \infty$$

It can be shown that the QR iteration is intimately related to the probably more familiar inverse iteration. We have

$$\begin{aligned} A_{k+1} &= Q_k^T Q_{k-1}^T \cdots Q_1^T A_1 Q_1 \cdots Q_{k-1} Q_k \\ &= P_k^T A_1 P_k \end{aligned}$$

Let

$$S_k = R_k \cdots R_1$$

then

$$\begin{aligned} P_k S_k &= P_{k-1} Q_k R_k S_{k-1} \\ &= P_{k-1} A_k S_{k-1} \end{aligned}$$

Noting that

$$A_1 P_{k-1} = P_{k-1} A_k$$

we get

$$\begin{aligned} P_k S_k &= A_1 P_{k-1} S_{k-1} \\ &= A^k \end{aligned}$$

Assume that A is nonsingular, then

$$P_k = A^{-k} S_k^T$$

Equating columns on both sides

$$P_k E = A^{-k} S_k^T E \tag{17}$$

where E consists of the last p columns of I.

Inverse iteration on p vectors can be defined as

$$AX_k = X_{k-1} L_k \qquad k = 1,2,... \tag{18}$$

where L_k is a lower triangular matrix chosen so that $X_k^T X_k = I$. It can be determined using the Gram-Schmidt process on the iteration vectors from the last vector to the first. Hence

$$X_k = A^{-k} X_o \bar{L}_k \quad ; \quad \bar{L}_k = L_1 \ldots L_k \tag{19}$$

Eq. (17) may be written as

$$P_k E = A^{-k} E \bar{S}_k \tag{20}$$

where $\bar{S}_k$ consists of the last p columns and rows of S_k^T. Using $X_k^T X_k = I$ and $(P_k E)^T (P_k E) = I$, we get

$$\bar{L}_k^{-T} \bar{L}_k^{-1} = X_o^T A^{-2k} X_o$$

and

$$\bar{S}_k^{-T} \bar{S}_k^{-1} = E^T A^{-2k} E$$

If we choose $X_o = E$ then $\bar{L}_k^{-1} = \bar{S}_k^{-1}$ because they are the Cholesky factors of the same positive definite matrix.

Considering Eqs. (19) and (20), we have shown that the inverse iteration algorithm defined in Eq. (18) if started with $X_o = E$ yields vectors X_k which are the last p columns in P_k of the QR algorithm.

In QR with shifts Eqs. (15) and (16) are

$$A_k - \mu_k I = Q_k R_k$$

$$A_{k+1} = R_k Q_k + \mu_k I \qquad k = 1,2,... \qquad A_1 = A$$

and the Rayleigh quotient iteration corresponds to μ_k equal to the (n,n) element in A_k.

In practice, QR is applied to tridiagonal matrices. Equation (15) rewritten is

$$Q_k^T A_k = R_k$$

where Q_k^T is obtained as a product of Jacobi rotation matrices which zero all subdiagonal elements in A_k . Ortega and Kaiser have developed explicit formulae which relate the elements in A_{k+1} to the elements in A_k [5]. Once the eigenvalues have been calculated to full machine precision, we calculate the eigenvectors of the tridiagonal matrix. Two steps of inverse iteration at shifts equal to the eigenvalues are sufficient to obtain the corresponding eigenvectors. These vectors need to be transformed with the Householder transformations used to obtain the eigenvectors of the original matrix.

APPENDIX II

COMPUTER PROGRAMS

SECANT

SSPACE

MODES

II.1 Program SECANT

The program uses the determinant search technique described in Chapter 3 to evaluate the smallest eigenvalues and corresponding vectors in the problem $Av = \lambda Bv$. Matrix A is assumed to be positive definite and B can be banded positive definite or diagonal non-negative definite.

The program is called with the parameters:

N = order of matrices A and B

NMAX = number of rows in storage blocks of matrices A and B

MA = half bandwidth of A including diagonal

MB = half bandwidth of B including diagonal

NROOT = number of required eigenvalues

NC = storage is provided for NC eigenvalues and vectors. Assume, for example, that the last eigenvalue lies in a cluster. Then the program may want to calculate it together with other close eigenvalues. The program stops when no more storage for eigenvalues and vectors is available.

Storage:

A(NMAX,MA) = matrix A

B(NMAX,MB) = matrix B

VV(N,NC) = eigenvectors

ROOT(NC) = eigenvalues

TIM(NC) = iteration times used

NITE(NC) = number of iterations

ERRVL(NC) = lower bounds on eigenvalues

ERRVR(NC) = upper bounds on eigenvalues

WW(N,NC), V(N), W(N) and MAXA(N) are working arrays.

The total storage required is

$$N*(MA + MB + 2*NC + 3) + 5*NC$$

One working tape is used. The program calls the following subroutines:

BANDET - performs the triangular factorization at a shift, the calculation of the determinant and the vector iteration.

MULT - carries out array multiplications.

The tolerances used are set up for the CDC 6400 with a 48 bit mantissa in floating point arithmetic. The eigenvalues are calculated to about 12 digit precision. Note that the logical variable ERRBD needs to be set .FALSE. if B is diagonal with zero elements, and that in this case program SECANTD is slightly more efficient (see Appendix II.3).

```
SECA   1          SUBROUTINE SECANT (A,B,V,MAXA,M,VV,WW,ROOT,TIM,ERRVL,ERRVR,
SECA   2         1NITE,N,NMAX,MA,MB,NROOT,NC)
SECA   3  C
SECA   4  C    PROGRAM TO COMPUTE SMALLEST EIGENVALUES AND ASSOCIATED VECTORS IN
SECA   5  C    THE GENERALIZED EIGENVALUE PROBLEM
SECA   6  C                          A*V=RT*B*V  (A POS DEF,B NONNEG DEF)
SECA   7  C
SECA   8  C                                  PROGRAMMED  BY  K.J.  BATHE
SECA   9  C                                  SESM  UNIV OF CALIF  BERKELEY  1971
SECA  10  C
SECA  11          DIMENSION A(NMAX,MA),B(NMAX,MB),V(1),W(1),VV(N,1),WW(N,1),
SECA  12         1ROOT(1),TIM(1),ERRVL(1),ERRVR(1)
SECA  13          INTEGER NITE(1),MAXA(1)
SECA  14          LOGICAL ERRBD
SECA  15  C
SECA  16  C    IF B IS POS DEF WE CAN SET ERRBD=.TRUE. AND ESTIMATE MORE ACCURATELY
SECA  17  C    A LOWER BOUND ON THE FIRST ROOT
SECA  18          ERRBD=.TRUE.
SECA  19  C
SECA  20  C     FOLLOWING TOLERANCES ARE SET FOR THE CDC 6400
SECA  21          ACTOL=1.0E-04
SECA  22          RCBTOL=1.0E-06
SECA  23          RTOL=1.0E-10
SECA  24          RQTOL=1.0E-12
SECA  25          SCALE=2.0**900
SECA  26  C
SECA  27          ITEM=10
SECA  28          NITEM=40
SECA  29  C
SECA  30  C    COMPACT MATRICES AND STORE
SECA  31          DO 10 J=2,MA
SECA  32          L=N*(J-1)
SECA  33          DO 10 I=1,N
SECA  34   10      A(L+I)=A(I,J)
SECA  35          IF (MB.EQ.1) GO TO 40
SECA  36          DO 20 J=2,MB
SECA  37          L=N*(J-1)
SECA  38          DO 20 I=1,N
SECA  39   20      B(L+I)=B(I,J)
SECA  40  C
SECA  41   40      REWIND 1
SECA  42          WRITE (1) A
SECA  43          WRITE (1) B
SECA  44  C
SECA  45          ETA=2.0
SECA  46          NOV=0
SECA  47          JR=1
SECA  48          NSK=0
SECA  49          NWA=N*MA
SECA  50          NWB=N*MB
SECA  51          NTF=5
SECA  52          ISC=1000
SECA  53  C
SECA  54          CALL SECOND (TIME1)
SECA  55          RA=0.0
SECA  56          RB=0.0
SECA  57          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,1)
SECA  58          FA=DETA
SECA  59          FB=FA
SECA  60          DETB=DETA
SECA  61  C
SECA  62  C    FIND LOWER BOUND ON SMALLEST EIGENVALUE
SECA  63          PRINT 1010
SECA  64          DO 100 I=1,N
SECA  65   100     V(I)=1.0
SECA  66          ITTB=0
SECA  67          RT=0.0
SECA  68          CALL MULT (W,B,V,N,MB)
SECA  69          KK=4
SECA  70   110     ITE=ITE+1
SECA  71          DO 120 I=1,N
SECA  72   120     V(I)=W(I)
SECA  73          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,KK)
SECA  74          KK=3
SECA  75          RQT=0.0
SECA  76          DO 130 I=1,N
SECA  77   130     RQT=RQT+W(I)*V(I)
SECA  78          CALL MULT (W,B,V,N,MB)
SECA  79          RQB=0.0
SECA  80          DO 140 I=1,N
SECA  81   140     RQB=RQB+W(I)*V(I)
SECA  82          RQ=RQT/RQB
SECA  83          PRINT 1006,RQ
SECA  84          RB=SQRT(RQB)
SECA  85          TOL=ABS(RQ-RT)/RQ
SECA  86          IF (TOL.LT.RCBTOL) GO TO 150
SECA  87          DO 160 I=1,N
SECA  88   160     W(I)=W(I)/RB
SECA  89          RT=RQ
SECA  90          IF (ITTE.LT.ITEM) GO TO 110
SECA  91          RB=RQ*(1.0-AMIN1(0.1,TOL))
SECA  92          GO TO 225
SECA  93   150     IF (ERRBD) GO TO 155
SECA  94          RB=RQ*(1.0-0.01)
SECA  95          GO TO 225
SECA  96  C
SECA  97   155     DO 170 I=1,N
SECA  98   170     V(I)=V(I)/RB
SECA  99          REWIND 1
SECA 100          READ (1) A
SECA 101          DO 200 I=1,NWB
SECA 102   200     A(I)=A(I)-RQ*B(I)
SECA 103          CALL MULT (W,A,V,N,MA)
SECA 104          DO 210 I=1,N
SECA 105   210     V(I)=W(I)
SECA 106          CALL BANDET (B,A,V,MAXA,N,NWB,NWA,RA,NSCH,DETA,ISC,2)
SECA 107          READ (1) B
SECA 108          ERR=0.0
SECA 109          DO 220 I=1,N
SECA 110   220     ERR=ERR+V(I)*W(I)
SECA 111          RB=RQ-SQRT(ERR)
SECA 112   225     IS=0
SECA 113   230     CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RB,NSCH,DETB,ISC,1)
SECA 114          PRINT 1020,RB,NSCH
SECA 115          FB=DETB
SECA 116          IF (NSCH.EQ.0) GO TO 300
SECA 117          IS=IS+1
SECA 118          IF (IS.LE.NTF) GO TO 240
SECA 119          PRINT 1030
SECA 120          STOP
SECA 121   240     RB=RB/(NSCH+1)
SECA 122          GO TO 230
SECA 123  C
SECA 124  C    ITERN FOR INDIVIDUAL ROOTS
SECA 125   300     PRINT 1040
SECA 126          NITE(JR)=1
SECA 127          PRINT 1050,JR,NITE(JR),RA,DETA,FA,ETA,ISC
SECA 128          NITE(JR)=2
SECA 129          PRINT 1050,JR,NITE(JR),RB,DETB,FB,ETA,ISC
SECA 130  C
SECA 131  C    WE STOP WHEN WE HAVE THE REQUIRED NO OF ROOTS SMALLER THAN RC AND
SECA 132  C    NOV=0
SECA 133   310     IF (NSCH.GE.NROOT) GO TO 900
SECA 134  C
SECA 135          DIF=FB-FA
SECA 136          IF (DIF.NE.0.0) GO TO 320
SECA 137          PRINT 1060
SECA 138          GO TO 900
SECA 139   320     DEL=FB*(RB-RA)/DIF
SECA 140          RC=RB-ETA*DEL
SECA 141          TOL=RCBTOL*RC
SECA 142          IF (ABS(RC-RB).GT.TOL) GO TO 330
SECA 143          PRINT 1070
SECA 144          ROOT(JR)=RC
SECA 145          GO TO 400
SECA 146  C
SECA 147   330     CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RC,NSCH,DETC,ISC,1)
SECA 148          FC=DETC
SECA 149          NITE(JR)=NITE(JR)+1
SECA 150          IF (FC.EQ.0.0) GO TO 340
```

```
SECA 151          JJ=JR-1
SECA 152          DO 350 K=1,JJ
SECA 153   350    FC=FC/(RC-ROOT(K))
SECA 154   340    PRINT 1050,JR,NITE(JR),RC,DETC,FC,ETA,ISC
SECA 155  C
SECA 156  C    IF WE HAVE MORE SIGNCHANGES THAN EIGENVALUES SMALLER THAN RC WE
SECA 157  C    START INV. ITERATION
SECA 158          NES=0
SECA 159          IF (JR.EQ.1) GO TO 390
SECA 160          DO 360 I=1,JJ
SECA 161   360    IF (ROOT(I).LT.RC) NES=NES+1
SECA 162   380    NOV=NSCH-NES
SECA 163          IF (NOV.EQ.0) GO TO 370
SECA 164          PRINT 1080,NOV
SECA 165          RCOT(JR)=RC
SECA 166          IF (NOV.GT.1) NSK=1
SECA 167  C
SECA 168          GO TO 400
SECA 169   370    RR=RA
SECA 170          FR=FA
SECA 171          DETR=DETA
SECA 172          RA=RB
SECA 173          FA=FB
SECA 174          DETA=DETB
SECA 175          RB=RC
SECA 176          FB=FC
SECA 177          DETB=DETC
SECA 178  C
SECA 179  C    WE RESET ETA IF NECESSARY
SECA 180          TOL=RB*ACTOL
SECA 181          IF (ABS(RA-RB).LT.TOL) ETA=ETA*2
SECA 182          IF (NITE(JR).LE.NITEM) GO TO 310
SECA 183          PRINT 1015,NITE(JR),JR
SECA 184          GO TO 900
SECA 185  C
SECA 186  C
SECA 187  C    CHECK FOR STORAGE
SECA 188   400    IF (JR.LE.NC) GO TO 405
SECA 189          PRINT 1090
SECA 190          GO TO 900
SECA 191  C
SECA 192   405    NOR=JR-1
SECA 193          CALL SECOND (TIM3)
SECA 194          PRINT 1100,NOR
SECA 195          IF (JR.EQ.1) GO TO 410
SECA 196          DO 420 I=1,N
SECA 197   420    V(I)=1.0
SECA 198          KK=4
SECA 199   410    CALL MULT (W,B,V,N,MB)
SECA 200          IS=0
SECA 201          RTA=0.0
SECA 202          GO TO 510
SECA 203  C
SECA 204  C    INVERSE ITERN
SECA 205   440    NITE(JR)=NITE(JR)+1
SECA 206          DO 450 I=1,N
SECA 207   450    V(I)=W(I)
SECA 208          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,KK)
SECA 209          IF (IS.EQ.1) GO TO 460
SECA 210          KK=3
SECA 211          RQT=0.0
SECA 212          DO 470 I=1,N
SECA 213   470    RQT=RQT+W(I)*V(I)
SECA 214          CALL MULT (W,B,V,N,MB)
SECA 215          RQB=0.0
SECA 216          DO 480 I=1,N
SECA 217   480    RQB=RQB+W(I)*V(I)
SECA 218          RQ=RQT/RQB
SECA 219          RT=RCOT(JR)+RQ
SECA 220          PRINT 1110,JR,NITE(JR),RT,RQ
SECA 221          TOL=RT*RQTOL
SECA 222          IF (ABS(RT-RTA).GT.TOL) GO TO 510
SECA 223          IS=1
SECA 224          GO TO 440
SECA 225  C
SECA 226   510    RTA=RT
SECA 227          BS=SQRT(RQB)
SECA 228          DO 490 I=1,N
SECA 229   490    W(I)=W(I)/BS
SECA 230          IF (NOV.EQ.0) GO TO 550
SECA 231          DO 520 K=1,NOV
SECA 232          AL=0.0
SECA 233          DO 530 I=1,N
SECA 234   530    AL=AL+VV(I,K)*W(I)
SECA 235          DO 540 I=1,N
SECA 236   540    W(I)=W(I)-AL*WW(I,K)
SECA 237   520    CONTINUE
SECA 238  C
SECA 239   550    IF (NITE(JR).LE.NITEM) GO TO 440
SECA 240          PRINT 1015,NITE(JR),JR
SECA 241          GO TO 900
SECA 242  C
SECA 243   460    RQT=0.0
SECA 244          DO 570 I=1,N
SECA 245   570    RQT=RQT+V(I)*W(I)
SECA 246          ERRT=RQB
SECA 247          CALL MULT (W,B,V,N,MB)
SECA 248          RQB=0.0
SECA 249          DO 580 I=1,N
SECA 250   580    RQB=RQB+V(I)*W(I)
SECA 251  C
SECA 252  C    OBTAIN A RATHER LARGE ERROR BOUND
SECA 253          RQ=RQT/RQB
SECA 254          RCOT(JR)=RCOT(JR)+RQ
SECA 255          ERR=SQRT(ERRT/RQB)
SECA 256          ERRVL(JR)=ROOT(JR)-ERR
SECA 257          ERRVR(JR)=ROOT(JR)+ERR
SECA 258  C
SECA 259          BS=SQRT(RQB)
SECA 260          DO 590 I=1,N
SECA 261          W(I)=W(I)/BS
SECA 262   590    V(I)=V(I)/BS
SECA 263          DO 600 I=1,N
SECA 264          WW(I,JR)=W(I)
SECA 265   600    VV(I,JR)=V(I)
SECA 266  C
SECA 267          CALL SECOND (TIM2)
SECA 268          TIM3=TIM2-TIM3
SECA 269          PRINT 1120,TIM3
SECA 270          TIM(JR)=TIM2-TIM1
SECA 271          TIM1=TIM2
SECA 272  C
SECA 273  C    DECIDE STRATEGY FOR ITERN TOWARDS NEXT ROOT
SECA 274          TOL=RTOL*ROOT(JR)
SECA 275          IF (NOV.GT.0) GO TO 700
SECA 276          IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 710
SECA 277          IF (RA.GT.C.0) GO TO 720
SECA 278          RA=RB/2.
SECA 279          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,1)
SECA 280          FA=DETA
SECA 281   720    RB=RA
SECA 282          FB=FA
SECA 283          DETB=DETA
SECA 284          RA=RR
SECA 285          FA=FR
SECA 286          DETA=DETR
SECA 287          GO TO 710
SECA 288  C
SECA 289   700    IF (ROOT(JR).GT.RC) NSK=1
SECA 290          IF (NSK.EQ.1) GO TO 730
SECA 291          IF (ABS(RC-ROOT(JR)).LT.TOL) GO TO 740
SECA 292          IF (ABS(ROOT(JR)-RB).LT.TOL) GO TO 750
SECA 293          RA=RB
SECA 294          FA=FB
SECA 295          DETA=DETB
SECA 296   750    RB=RC
SECA 297          FB=FC
SECA 298          DETB=DETC
SECA 299          GO TO 710
SECA 300   740    IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 710
```

```
SECA 301          IF (RA.GT.C.0) GO TO 760
SECA 302          RA=RB/2.
SECA 303          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,I)
SECA 304          FA=DETA
SECA 305    760   RB=RA
SECA 306          FB=FA
SECA 307          DETB=DETA
SECA 308          RA=RR
SECA 309          FA=FR
SECA 310          DETA=DETR
SECA 311    710   FA=FA/(RA-ROOT(JR))
SECA 312          FB=FB/(RB-ROOT(JR))
SECA 313          JR=JR+1
SECA 314          ETA=2.0
SECA 315          GO TO 300
SECA 316  C
SECA 317    730   IF (RA.GT.0.0) GO TO 780
SECA 318          RA=RB/2.
SECA 319          CALL BANDET (A,B,V,MAXA,N,NWA,NWB,RA,NSCH,DETA,ISC,I)
SECA 320          FA=DETA
SECA 321    780   IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 770
SECA 322          RB=RA
SECA 323          FB=FA
SECA 324          DETB=DETA
SECA 325          RA=RR
SECA 326          FA=FR
SECA 327          DETA=DETR
SECA 328    770   FA=FA/(RA-ROOT(JR))
SECA 329          FB=FB/(RB-ROOT(JR))
SECA 330          FR=FR/(RR-ROOT(JR))
SECA 331          IF (ROOT(JR).LE.RC) NOV=NOV-1
SECA 332          JR=JR+1
SECA 333          NITE(JR)=0
SECA 334          ROOT(JR)=RC
SECA 335          IF (NOV.GT.0) GO TO 400
SECA 336          NSK=0
SECA 337          ETA=2.0
SECA 338          GO TO 300
SECA 339  C
SECA 340    900   NROOT=JR-1
SECA 341          IF (NROOT.EQ.0) RETURN
SECA 342          PRINT 1130
SECA 343          PRINT 1004,(ROOT(J),J=1,NROOT)
SECA 344          PRINT 1140
SECA 345          PRINT 1006,(NITE(J),J=1,NROOT)
SECA 346          PRINT 1150
SECA 347          PRINT 1008,(TIM(J),J=1,NROOT)
SECA 348          PRINT 1160
SECA 349          PRINT 1004,(ERRVL(J),J=1,NROOT)
SECA 350          PRINT 1004,(ERRVR(J),J=1,NROOT)
SECA 351  C
SECA 352  C   APRANGE EIGENVALUES AND VECTORS IN ASCENDING ORDER
SECA 353          IF (JR.EQ.2) GO TO 950
SECA 354          JR=JR-2
SECA 355    910   IS=0
SECA 356          DO 920 I=1,JR
SECA 357          IF (ROOT(I+1).GE.ROOT(I)) GO TO 920
SECA 358          IS=IS+1
SECA 359          RT=ROOT(I+1)
SECA 360          ROOT(I+1)=ROOT(I)
SECA 361          ROOT(I)=RT
SECA 362          DO 930 K=1,N
SECA 363          RT=VV(K,I+1)
SECA 364          VV(K,I+1)=VV(K,I)
SECA 365    930   VV(K,I)=RT
SECA 366    920   CONTINUE
SECA 367          IF (IS.GT.0) GO TO 910
SECA 368  C
SECA 369    950   PRINT 1170
SECA 370          NROOT=NSCH
SECA 371          PRINT 1130
SECA 372          PRINT 1004,(ROOT(J),J=1,NROOT)
SECA 373          PRINT 1180
SECA 374          DO 960 J=1,NROOT
SECA 375    960   PRINT 1002,(VV(I,J),I=1,N)
SECA 376          RETURN
SECA 377  C
SECA 378   1002   FORMAT (1H ,12E11.4)
SECA 379   1004   FORMAT (1HC,8E20.12)
SECA 380   1006   FORMAT (1HC,6I20)
SECA 381   1008   FORMAT (1HC,6F20.2)
SECA 382   1010   FORMAT (1H1,63HINVERSE ITERN GIVES FOLLOWING APPROXIMATN TO LOWES
SECA 383       1T EIGENVALUE  )
SECA 384   1015   FORMAT (41HOWE ABANDON ITERN BECAUSE NO OF ITERN IS I3,9H FOR ROO
SECA 385       1T I3 )
SECA 386   1020   FORMAT (5HORB = E20.12,7H NSCH = I4)
SECA 387   1030   FORMAT (30HOWE BETTER CHECK THE MATRICES  )
SECA 388   1040   FORMAT (1H1,4X,4HROOT,4X,4HNITE,18X,2HRC,15X,12HDET (A-RC**B),15X,
SECA 389      /2HEC,13X,3HETA,4X,3HISC)
SECA 390   1050   FORMAT (1HC,4X,I4,4X,I4,8X,3E22.14,F7.2,I6)
SECA 391   1060   FORMAT (42HOTHE DEFLATED POLYNOMIAL HAS NO MORE ROOTS  )
SECA 392   1070   FORMAT (29HO(RC-RB) IS SMALLER THAN TOL )
SECA 393   1080   FORMAT (16HOWE JUMPED OVER I4,16H UNKNOWN ROOT(S)  )
SECA 394   1090   FORMAT (1H1,36HNO MORE STORAGE FOR VECTORS  WE QUIT  )
SECA 395   1100   FORMAT (1HC,34X,4HROOT,18X,2HRW,18X,4HNOR=,I2)
SECA 396   1110   FORMAT (1HC,4X,I4,4X,I4,8X,2E22.14)
SECA 397   1120   FORMAT (20HOTIME FOR INV ITERN F5.2)
SECA 398   1130   FORMAT (20HOTHE EIGENVALUES ARE  /)
SECA 399   1140   FORMAT (42HONO OF ITERATIONS FOR EACH EIGENVALUE ARE  /)
SECA 400   1150   FORMAT (30HOTIME USED FOR EACH EIGENVALUE /)
SECA 401   1160   FORMAT (43HOFOLLOWING ARE ERROR BOUNDS ON EIGENVALUES )
SECA 402   1170   FORMAT (1H1,62HWE ACCEPT FOLLOWING EIGENVALUES AND VECTORS ARRANG
SECA 403       1ED IN ORDER )
SECA 404   1180   FORMAT (22HOTHE EIGENVECTORS ARE  //)
SECA 405  C
SECA 406          END
```

```
MULT   1          SUBROUTINE MULT (W,A,V,NN,MA)
MULT   2  C
MULT   3          DIMENSION A(1),W(1),V(1)
MULT   4  C
MULT   5          NM=NN*(MA-1)
MULT   6          NMA=NN-MA+1
MULT   7          DO 20 I=1,NN
MULT   8          W(I)=0.0
MULT   9          K=I-1
MULT  10          IF (NMA-1) 10,15,15
MULT  11    10    NM=NM-NN
MULT  12    15    IL=NM+I
MULT  13          DO 20 J=1,IL,NN
MULT  14          K=K+1
MULT  15    20    W(I)=W(I)+A(J)*V(K)
MULT  16  C
MULT  17          IF (MA-1) 30,100,30
MULT  18  C
MULT  19    30    KK=NN
MULT  20          DO 40 I=2,MA
MULT  21          II=I-1
MULT  22          KK=KK+NN
MULT  23          KJ=KK
MULT  24          DO 40 J=1,II
MULT  25          KJ=KJ-NN
MULT  26    40    W(I)=W(I)+A(KJ+J)*V(J)
MULT  27          MA1=MA+1
MULT  28          IJ=1
MULT  29          DO 50 I=MA1,NN
MULT  30          KJ=KK
MULT  31          IJ=IJ+1
MULT  32          II=I-1
MULT  33          DO 50 J=IJ,II
MULT  34          KJ=KJ-NN
MULT  35    50    W(I)=W(I)+A(KJ+J)*V(J)
MULT  36  C
MULT  37   100    RETURN
```

```
MULT  38          END
```

```
BAND   1          SUBROUTINE BANDET (A,B,V,MAXA,NN,NWA,NWB,RA,NSCH,DET,ISCALE,KK)
BAND   2  C
BAND   3          DIMENSION A(NWA),B(NWB),V(1),MAXA(1)
BAND   4  C
BAND   5          NR=NN-1
BAND   6          IF (KK-3) 100,700,800
BAND   7  C
BAND   8   100    TOL=1.CE+07
BAND   9          RTOL=1.CE-10
BAND  10          SCALE=2.C**900
BAND  11          NTF=3
BAND  12          IS=1
BAND  13   120    IF (RA.EQ.C.0) GO TO 160
BAND  14          REWIND 1
BAND  15          READ (1) A
BAND  16          DO 140 I=1,NWB
BAND  17   140    A(I)=A(I)-RA*B(I)
BAND  18   160    IF (NWA.EQ.NN) GO TO 230
BAND  19          DO 200 N=1,NR
BAND  20          IH=N+NWA-NN
BAND  21   210    IF (A(IH)) 220,215,220
BAND  22   215    IH=IH-NN
BAND  23          GO TO 210
BAND  24   220    MAXA(N)=IH
BAND  25          PIV=A(N)
BAND  26          IF (PIV) 221,225,221
BAND  27   221    IL=N+NN
BAND  28          L=N
BAND  29          DO 240 I=IL,IH,NN
BAND  30          L=L+1
BAND  31          C=A(I)
BAND  32          IF (C) 225,240,225
BAND  33   225    C=C/PIV
BAND  34          IF (ABS(C).LT.TOL) GO TO 235
BAND  35   226    IS=IS+1
BAND  36          IF (IS.LE.NTF) GO TO 245
BAND  37          PRINT 1000,NTF
BAND  38          STOP
BAND  39   245    RA=RA*(1.0-RTOL)
BAND  40          GO TO 120
BAND  41   235    J=L-I
BAND  42          DO 260 K=I,IH,NN
BAND  43   260    A(K+J)=A(K+J)-C*A(K)
BAND  44          A(I)=C
BAND  45   240    CONTINUE
BAND  46   200    CONTINUE
BAND  47   230    IF (A(NN).NE.0.0) GO TO 280
BAND  48          AA=ABS(A(1))
BAND  49          DO 290 I=2,NR
BAND  50   290    AA=AA+ABS(A(I))
BAND  51          A(NN)=-(AA/NR)*1.CE-16
BAND  52   280    IF (KK.EQ.2) GO TO 700
BAND  53  C
BAND  54          NSCH=0
BAND  55          ISC=0
BAND  56          DET=1.0
BAND  57          DO 300 I=1,NN
BAND  58          IF (ABS(DET).LT.SCALE) GO TO 320
BAND  59          DET=DET/SCALE
BAND  60          ISC=ISC+1
BAND  61   320    DET=DET*A(I)
BAND  62   300    IF (A(I).LT.0.) NSCH=NSCH+1
BAND  63  C
BAND  64          IF (ISCALE.LT.1000) GO TO 340
BAND  65          ISCALE=ISC
BAND  66          GO TO 900
BAND  67   340    IF (ISC-ISCALE) 350,900,370
BAND  68   350    DET=DET/SCALE
BAND  69          GO TO 900
BAND  70   370    DET=DET*SCALE
BAND  71          GO TO 900
BAND  72  C
BAND  73   700    IL=NN
BAND  74          DO 400 N=1,NR
BAND  75          C=V(N)
BAND  76          V(N)=C/A(N)
BAND  77          IF (NWA-NN) 410,400,410
BAND  78   410    IL=IL+1
BAND  79          IH=MAXA(N)
BAND  80          K=N
BAND  81          DO 420 I=IL,IH,NN
BAND  82          K=K+1
BAND  83   420    V(K)=V(K)-C*A(I)
BAND  84   400    CONTINUE
BAND  85          V(NN)=V(NN)/A(NN)
BAND  86  C
BAND  87   800    IF (NWA-NN) 430,900,430
BAND  88   430    N=NN
BAND  89          DO 440 L=2,NN
BAND  90          N=N-1
BAND  91          IL=N+NN
BAND  92          IH=MAXA(N)
BAND  93          K=N
BAND  94          DO 460 I=IL,IH,NN
BAND  95          K=K+1
BAND  96   460    V(N)=V(N)-A(I)*V(K)
BAND  97   440    CONTINUE
BAND  98   900    RETURN
BAND  99   1000   FORMAT (1H1,20H TRIANG FACTORIZATN I3,32H TIMES ABANDONED,CHECK
BAND 100         1MATRICES  )
BAND 101          END
```

II.2 Program SSPACE

Program SSPACE uses the subspace iteration technique described in Chapter 4 to calculate the smallest eigenvalues and corresponding vectors in the problem $Av = \lambda Bv$. Matrix A is assumed to be positive definite and B can be banded positive definite or diagonal non-negative definite.

The program is called with the parameters:

N = order of matrices A and B

NMAX = number of rows in storage blocks of matrices A and B

MA = half bandwidth of A including diagonal

MB = half bandwidth of B including diagonal

NROOT = number of required eigenvalues

NC = min {2*NROOT, NROOT + 8} and is initialized in the program

Storage:

A(NMAX,MA) = matrix A

B(NMAX,MB) = matrix B

T(N,NC) = eigenvectors

EIGV(NC) = eigenvalues

R(N,NC), W(N), MAXA(N), TT(N), AR(NC,NC), BR(NC,NC), VEC(NC,NC), RTOLV(NC) and D(NC) are working arrays.

The total storage required is

N*(MA + MB + 2*NC + 3) + NC*(3*NC + 3)

Two working tapes are used. The program calls the following subroutines:

BAN - performs the triangular factorizations and vector iterations

JACOBI - solves the small generalized eigenvalue problem (the listing is given with MODES).

IIGRAM - calls BAN to perform vector iterations and orthonormalizes the iteration vectors using the Gram-Schmidt process.

MULT - carries out array multiplications (the listing is given with SECANT).

ERRAN - calculates error bounds on the eigenvalues and performs Sturm Sequence checks. If an eigenvalue is missing, the program finds an interval in which the missing eigenvalue lies. ERRAN can only be called when B is positive definite and then the logical variable CHECK must be set .TRUE..

The tolerances used are set up for the CDC 6400 with a 48 bit mantissa in floating point arithmetic. Referring to the listing given and Table 1 the program performs the following sequence of subspace iterations: a,b,a,b,c,c,c.... This sequence is changed by adjusting the variables INVM, INCR, IIG and NGR. The eigenvalues are calculated to about 5 digit precision.

```
SSPA   1          SUBROUTINE SSPACE (A,B,T,R,TT,W,MAXA,AR,BR,VEC,EIGV,D,RTOLV,N,
SSPA   2         1NMAX,MA,MB,NROOT,NC)
SSPA   3  C
SSPA   4  C     PROGRAM TO COMPUTE SMALLEST EIGENVALUES AND ASSOCIATED VECTORS
SSPA   5  C     IN THE GENERALIZED EIGENVALUE PROBLEM
SSPA   6  C                          A*V=R*B*V  (A POS DEF,B NONNEG DEF)
SSPA   7  C
SSPA   8  C                                  PROGRAMMED  BY  K.J. BATHE
SSPA   9  C                                  SESM  UNIV OF CALIF  BERKELEY  1971
SSPA  10  C
SSPA  11          DIMENSION A(NMAX,MA),B(NMAX,MB),T(N,1),R(N,1),TT(1),W(1),EIGV(1),
SSPA  12         1D(1),VEC(NC,1),AR(NC,1),BR(NC,1),RTOLV(1)
SSPA  13          INTEGER MAXA(N)
SSPA  14          LOGICAL CHECK
SSPA  15  C
SSPA  16  C     IF B IS POS DEF WE CAN SET CHECK=.TRUE. TO CALCULATE 'EXACT' ERROR
SSPA  17  C     BOUNDS AND APPLY THE STURM SEQUENCE CHECK
SSPA  18          CHECK=.TRUE.
SSPA  19  C
SSPA  20          MB1=MB+1
SSPA  21          NWA=MA*N
SSPA  22          NWB=MB*N
SSPA  23  C
SSPA  24  C     FOLLOWING TOLERANCES ARE SET FOR THE CDC 6400
SSPA  25          RTOL=1.0E-06
SSPA  26          TOLJ=1.0E-12
SSPA  27  C
SSPA  28          INVM=0
SSPA  29          NCR=2
SSPA  30          INCR=0
SSPA  31          IIG=2
SSPA  32          NSCH=0
SSPA  33          NITEM=16
SSPA  34          NC=2*NROOT
SSPA  35          IF (NROOT.GT.8) NC=NROOT+8
SSPA  36          NC1=NC-1
SSPA  37  C
SSPA  38  C     COMPACT MATRICES AND STORE
SSPA  39          DO 10 J=2,MA
SSPA  40          L=N*(J-1)
SSPA  41          DO 10 I=1,N
SSPA  42    10    A(L+I)=A(I,J)
SSPA  43          IF (MB.EQ.1) GO TO 40
SSPA  44          DO 20 J=2,MB
SSPA  45          L=N*(J-1)
SSPA  46          DO 20 I=1,N
SSPA  47    20    B(L+I)=B(I,J)
SSPA  48  C
SSPA  49    40    REWIND 1
SSPA  50          WRITE (1) A
SSPA  51          WRITE (1) B
SSPA  52  C
SSPA  53  C     TRIANGULARIZE MATRIX A
SSPA  54          CALL SECOND (TIM1)
SSPA  55          SHIFT=0.0
SSPA  56          CALL BAN (A,B,TT,MAXA,N,NWA,NWB,SHIFT,NSCH,1)
SSPA  57  C
SSPA  58  C     ESTABLISH STARTING TRANSFORMATION VECTORS
SSPA  59          NC=N/NC
SSPA  60          DO 200 I=1,N
SSPA  61   200    W(I)=B(I)/A(I)
SSPA  62          DO 220 I=1,N
SSPA  63          R(I,1)=B(I)
SSPA  64          DO 220 J=2,NC
SSPA  65   220    R(I,J)=0.0
SSPA  66  C
SSPA  67          L=N-NC
SSPA  68          DO 240 J=2,NC
SSPA  69          RT=0.0
SSPA  70          DO 260 I=1,L
SSPA  71          IF (W(I).LT.RT) GO TO 260
SSPA  72          RT=W(I)
SSPA  73          IJ=I
SSPA  74   260    CONTINUE
SSPA  75          DO 280 I=L,N
SSPA  76          IF (W(I).LT.RT) GO TO 280
SSPA  77          RT=W(I)
SSPA  78          IJ=I
SSPA  79   280    CONTINUE
SSPA  80          W(IJ)=0.0
SSPA  81          L=L-NC
SSPA  82   240    R(IJ,J)=1.
SSPA  83  C
SSPA  84          DO 300 LL=1,NCR
SSPA  85          CALL IIGRAM (A,B,T,R,TT,W,MAXA,N,MA,MB1,NWB,IIG,NC)
SSPA  86   300    CONTINUE
SSPA  87  C
SSPA  88          DO 340 I=1,NC
SSPA  89   340    D(I)=0.0
SSPA  90          NITE=IIG*NCR
SSPA  91          INV=0
SSPA  92   400    NITE=NITE+1
SSPA  93  C
SSPA  94  C     PERFORM INV INVERSE ITERNS AND THEN  ITH USING GM-SCHMIDT
SSPA  95          IF ((INVM.EQ.0).OR.(NITE.EQ.(IIG*NCR+1))) GO TO 440
SSPA  96          INV=INV+INCR
SSPA  97          IF (INV.GT.INVM) INV=INVM
SSPA  98          CALL SECOND (TIM2)
SSPA  99          CALL IIGRAM (A,B,T,R,TT,W,MAXA,N,MA,NWA,NWB,INV,NC)
SSPA 100          CALL SECOND (TIM3)
SSPA 101          TIM3=TIM3-TIM2
SSPA 102          PRINT 1000,INV,TIM3
SSPA 103          NITE=NITE+INV
SSPA 104   440    PRINT 1010,NITE
SSPA 105  C
SSPA 106  C     FIND THE PROJECTIONS OF OPERATORS A AND B
SSPA 107          CALL SECOND (TIM4)
SSPA 108          DO 460 J=1,NC
SSPA 109          DO 470 K=1,N
SSPA 110   470    TT(K)=R(K,J)
SSPA 111          CALL BAN (A,B,TT,MAXA,N,NWA,NWB,SHIFT,NSCH,2)
SSPA 112          DO 480 I=J,NC
SSPA 113          ART=0.0
SSPA 114          DO 490 K=1,N
SSPA 115   490    ART=ART+R(K,I)*TT(K)
SSPA 116   480    AR(I,J)=ART
SSPA 117          DO 500 K=1,N
SSPA 118   500    R(K,J)=TT(K)
SSPA 119   460    CONTINUE
SSPA 120          DO 560 J=1,NC
SSPA 121          DO 570 K=1,N
SSPA 122   570    W(K)=R(K,J)
SSPA 123          CALL MULT (TT,B,W,N,MB)
SSPA 124          DO 580 I=J,NC
SSPA 125          BRT=0.0
SSPA 126          DO 590 K=1,N
SSPA 127   590    BRT=BRT+R(K,I)*TT(K)
SSPA 128   580    BR(I,J)=BRT
SSPA 129          DO 600 K=1,N
SSPA 130   600    T(K,J)=TT(K)
SSPA 131   560    CONTINUE
SSPA 132          DO 610 J=1,NC
SSPA 133          DO 610 I=J,NC
SSPA 134          AR(J,I)=AR(I,J)
SSPA 135   610    BR(J,I)=BR(I,J)
SSPA 136  C
SSPA 137  C     SOLVE FOR EIGENSYSTEM OF SUBSPACE OPERATORS
SSPA 138          PRINT 1020
SSPA 139          DO 620 I=1,NC
SSPA 140   620    PRINT 1005,(AR(I,J),J=1,NC)
SSPA 141          PRINT 1030
SSPA 142          DO 630 I=1,NC
SSPA 143   630    PRINT 1005,(BR(I,J),J=1,NC)
SSPA 144  C
SSPA 145          CALL SECOND (TIM5)
SSPA 146          CALL JACOBI (AR,BR,VEC,EIGV,W,NC,TOLJ)
SSPA 147          DO 655 J=1,NC
SSPA 148          RT=SQRT(BR(J,J))
SSPA 149          DO 655 K=1,NC
SSPA 150   655    VEC(K,J)=VEC(K,J)/RT
```

```
SSPA 191          CALL SECOND (TIM6)
SSPA 192  C
SSPA 193          PRINT 1040
SSPA 194          PRINT 1020
SSPA 195          DO 640 I=1,NC
SSPA 196   640     PRINT 1005,(AR(I,J),J=1,NC)
SSPA 197          PRINT 1030
SSPA 198          DO 650 I=1,NC
SSPA 199   650     PRINT 1005,(BR(I,J),J=1,NC)
SSPA 200  C
SSPA 201  C     ARRANGE EIGENVALUES IN ASCENDING ORDER
SSPA 202   690     IS=0
SSPA 203          DO 700 I=1,NC1
SSPA 204          IF (EIGV(I+1).GE.EIGV(I)) GO TO 700
SSPA 205          IS=IS+1
SSPA 206          EIGVT=EIGV(I+1)
SSPA 207          EIGV(I+1)=EIGV(I)
SSPA 208          EIGV(I)=EIGVT
SSPA 209          BT=BR(I+1,I+1)
SSPA 210          BR(I+1,I+1)=BR(I,I)
SSPA 211          BR(I,I)=BT
SSPA 212          DO 710 K=1,NC
SSPA 213          RT=VEC(K,I+1)
SSPA 214          VEC(K,I+1)=VEC(K,I)
SSPA 215   710     VEC(K,I)=RT
SSPA 216   700     CONTINUE
SSPA 217          IF (IS.GT.0) GO TO 690
SSPA 218          PRINT 1035
SSPA 219          PRINT 1006,(EIGV(I),I=1,NC)
SSPA 220  C
SSPA 221  C     CHECK FOR CONVERGENCE OF EIGENVALUES
SSPA 222          DO 660 I=1,NC
SSPA 223          DIF=ABS(EIGV(I)-D(I))
SSPA 224   660     RTOLV(I)=DIF/EIGV(I)
SSPA 225          PRINT 1050
SSPA 226          PRINT 1005,(RTOLV(I),I=1,NC)
SSPA 227  C
SSPA 228          DO 670 I=1,NROOT
SSPA 229          IF (RTOLV(I).GT.RTOL) GO TO 680
SSPA 230   670     CONTINUE
SSPA 231          PRINT 1060,RTOL
SSPA 232          GO TO 800
SSPA 233  C
SSPA 234   680     IF (NITE.LT.NITEM) GO TO 695
SSPA 235          PRINT 1070
SSPA 236          GO TO 800
SSPA 237  C
SSPA 238  C     CALCULATE B TIMES APPROX EIGENVECTORS
SSPA 239   695     DO 720 J=1,NC
SSPA 240          DO 720 I=1,N
SSPA 241          RT=0.0
SSPA 242          DO 730 K=1,NC
SSPA 243   730     RT=RT+T(I,K)*VEC(K,J)
SSPA 244   720     P(I,J)=RT
SSPA 245  C
SSPA 246          DO 740 I=1,NC
SSPA 247   740     D(I)=EIGV(I)
SSPA 248          CALL SECOND (TIM7)
SSPA 249          TIM6=TIM6-TIM5
SSPA 250          TIM7=TIM7-TIM4
SSPA 251          PRINT 1080,TIM6
SSPA 252          PRINT 1090,TIM7
SSPA 253          GO TO 400
SSPA 254  C
SSPA 255   800     CALL SECOND (TIM7)
SSPA 256          TIM6=TIM6-TIM5
SSPA 257          TIM7=TIM7-TIM4
SSPA 258          PRINT 1080,TIM6
SSPA 259          PRINT 1090,TIM7
SSPA 260  C
SSPA 261  C     ORTHONORMALIZE VECTORS
SSPA 262          DO 810 J=1,NC
SSPA 263          DO 810 I=1,N
SSPA 264          RT=0.0
SSPA 265          DO 820 K=1,NC
SSPA 266   820     RT=RT+P(I,K)*VEC(K,J)
SSPA 267   810     T(I,J)=RT
SSPA 268          PRINT 1100
SSPA 269          PRINT 1006,(EIGV(I),I=1,NC)
SSPA 270          PRINT 1110
SSPA 271          DO 840 J=1,NC
SSPA 272   840     PRINT 1005,(T(K,J),K=1,N)
SSPA 273  C
SSPA 274          IF (CHECK) GO TO 860
SSPA 275          CALL SECOND (TIM9)
SSPA 276          GO TO 900
SSPA 277  C
SSPA 278  C     CALCLUATE ERROR BOUNDS ON EIGENVALUES AND CHECK FOR MISSING ROOTS
SSPA 279   860     CALL ERRAN (A,B,T,W,TI,MAXA,EIGV,P,D,...,RTOLV,N,MA,NWA,NWB,
SSPA 280         INROOT,NC,SHIFT,TEND,NSCH)
SSPA 281          CALL SECOND (TIM9)
SSPA 282          TIM8=TIM9-TIM8
SSPA 283          PRINT 1120,TIM8
SSPA 284   900     TIM9=TIM9-TIM1
SSPA 285          PRINT 1130,TIM9
SSPA 286          RETURN
SSPA 287  C
SSPA 288   1005   FORMAT (1H ,12E11.4)
SSPA 289   1006   FORMAT (1HO,6F22.14)
SSPA 290   1000   FORMAT (1H1,14HTIME USED FOR I2,22H INV ITERNS AND ORTHO  F6.2)
SSPA 291   1010   FORMAT (1H1,12HITERATION NO I4)
SSPA 292   1020   FORMAT (10HOMATRIX AR  )
SSPA 293   1030   FORMAT (10HOMATRIX BR  )
SSPA 294   1035   FORMAT (30HOEIGENVALUES OF AR-LAMBDA*BR  )
SSPA 295   1040   FORMAT (40HOAR AND BR AFTER JACOBI DIAGONALIZATION  )
SSPA 296   1050   FORMAT (38HOREL TOLERANCE ON EIGENVALUES REACHED  )
SSPA 297   1060   FORMAT (30HOCONVERGENCE REACHED FOR RTOL  E10.4)
SSPA 298   1070   FORMAT (1H1,31HWE ACCEPT CURRENT ITERN VALUES  )
SSPA 299   1080   FORMAT (35HOTIME USED IN EIGENVALUE SOLUTION  F8.2)
SSPA 300   1090   FORMAT (30HOTIME USED IN ITERATION STEP  F13.2)
SSPA 301   1100   FORMAT (27HOTHE FINAL EIGENVALUES ARE  )
SSPA 302   1110   FORMAT (23HOTHE NORMALIZED EIGENVECTORS ARE  )
SSPA 303   1120   FORMAT (28HOTIME USED FOR FINAL CHECKS  F6.2)
SSPA 304   1130   FORMAT (17HOTOTAL TIME USED F6.2)
SSPA 305  C
SSPA 306          END
```

```
BAN    1          SUBROUTINE BAN (A,B,V,MAXA,NN,NWA,NWB,RA,NSCH,KK)
BAN    2  C
BAN    3          DIMENSION A(NWA),B(NWB),V(1),MAXA(1)
BAN    4  C
BAN    5          NR=NN-1
BAN    6          IF (KK-2) 100,700,800
BAN    7  C
BAN    8   100    TOL=1.0E+07
BAN    9          RTOL=1.0E-06
BAN   10          NFT=3
BAN   11          IS=1
BAN   12   120    IF (RA.EQ.C.0) GO TO 160
BAN   13          REWIND 1
BAN   14          READ (1) A
BAN   15          DO 140 I=1,NWB
BAN   16   140     A(I)=A(I)-RA*B(I)
BAN   17   160    IF (NWA.EQ.NN) GO TO 230
BAN   18          DO 200 N=1,NR
BAN   19          IH=N+NWA-NN
BAN   20   210    IF (A(IH)) 220,215,220
BAN   21   215     IH=IH-NN
BAN   22          GO TO 210
BAN   23   220    MAXA(N)=IH
BAN   24          PIV=A(N)
BAN   25          IF (PIV) 221,226,221
BAN   26   221    IL=N+NN
```

```
BAN    27          L=N
BAN    28          DO 240 I=IL,IH,NN
BAN    29          L=L+1
BAN    30          C=A(I)
BAN    31          IF (C) 225,240,225
BAN    32   225    C=C/PIV
BAN    33          IF (ABS(C).LT.TOL) GO TO 235
BAN    34   226    IS=IS+1
BAN    35          IF (IS.LE.NFT) GO TO 245
BAN    36          PRINT 1000,NFT
BAN    37          STOP
BAN    38   245    RA=RA*(1.-RTOL)
BAN    39          GO TO 120
BAN    40   235    J=L-I
BAN    41          DO 260 K=I,IH,NN
BAN    42   260    A(K+J)=A(K+J)-C*A(K)
BAN    43          A(I)=C
BAN    44   240    CONTINUE
BAN    45   200    CONTINUE
BAN    46   230    IF (A(NN).NE.0.0) GO TO 280
BAN    47          AA=ABS(A(1))
BAN    48          DO 290 I=2,NR
BAN    49   290    AA=AA+ABS(A(I))
BAN    50          A(NN)=-(AA/NR)*1.0E-16
BAN    51   C
BAN    52   280    NSCH=0
BAN    53          DO 300 I=1,NN
BAN    54   300    IF (A(I).LT.0.) NSCH=NSCH+1
BAN    55   C
BAN    56          GO TO 900
BAN    57   C
BAN    58   700    IL=NN
BAN    59          DO 400 N=1,NR
BAN    60          C=V(N)
BAN    61          V(N)=C/A(N)
BAN    62          IF (NWA-NN) 410,400,410
BAN    63   410    IL=IL+1
BAN    64          IH=MAXA(N)
BAN    65          K=N
BAN    66          DO 420 I=IL,IH,NN
BAN    67          K=K+1
BAN    68   420    V(K)=V(K)-C*A(I)
BAN    69   400    CONTINUE
BAN    70          V(NN)=V(NN)/A(NN)
BAN    71   C
BAN    72   800    IF (NWA-NN) 430,900,430
BAN    73   430    N=NN
BAN    74          DO 440 L=2,NN
BAN    75          N=N-1
BAN    76          IL=N+NN
BAN    77          IH=MAXA(N)
BAN    78          K=N
BAN    79          DO 460 I=IL,IH,NN
BAN    80          K=K+1
BAN    81   460    V(N)=V(N)-A(I)*V(K)
BAN    82   440    CONTINUE
BAN    83   900    RETURN
BAN    84   C
BAN    85   1000   FORMAT (1H1,20H TRIANG FACTORIZATN I3,32H TIMES ABONDONED,CHECK
BAN    86        1MATRICES  )
BAN    87          END
```

```
IIGR    1          SUBROUTINE IIGRAM (A,B,T,R,TT,W,MAXA,N,MB,NWA,NWB,INV,NIV)
IIGR    2   C
IIGR    3          DIMENSION A(NWA),B(NWB),R(N,1),T(N,1),TT(1),W(1)
IIGR    4          INTEGER MAXA(1)
IIGR    5   C
IIGR    6   C     INVERSE ITERN
IIGR    7          SHIFT=0.0
IIGR    8          DO 100 J=1,NIV
IIGR    9          DO 110 I=1,N
IIGR   10   110    TT(I)=R(I,J)
IIGR   11          DO 120 JK=1,INV
IIGR   12          CALL BAN (A,B,TT,MAXA,N,NWA,NWB,SHIFT,NSCH,2)
IIGR   13          CALL MULT (W,B,TT,N,MB)
IIGR   14          IF (JK.EQ.INV) GO TO 130
IIGR   15          DO 140 I=1,N
IIGR   16   140    TT(I)=W(I)
IIGR   17   120    CONTINUE
IIGR   18   130    DO 150 I=1,N
IIGR   19          T(I,J)=TT(I)
IIGR   20   150    R(I,J)=W(I)
IIGR   21   100    CONTINUE
IIGR   22   C
IIGR   23   C     GRAM-SCHMIDT ORTHONORMALIZATION
IIGR   24          BS=0.0
IIGR   25          DO 200 K=1,N
IIGR   26   200    BS=BS+T(K,1)*R(K,1)
IIGR   27          BS=SQRT(BS)
IIGR   28          DO 220 K=1,N
IIGR   29          T(K,1)=T(K,1)/BS
IIGR   30   220    R(K,1)=R(K,1)/BS
IIGR   31          IF (NIV.EQ.1) RETURN
IIGR   32          DO 300 J=2,NIV
IIGR   33          JK=J-1
IIGR   34   310    DO 320 JJ=1,JK
IIGR   35          AL=0.0
IIGR   36          DO 340 K=1,N
IIGR   37   340    AL=AL+T(K,J)*R(K,JJ)
IIGR   38          DO 360 K=1,N
IIGR   39          T(K,J)=T(K,J)-AL*T(K,JJ)
IIGR   40   360    R(K,J)=R(K,J)-AL*R(K,JJ)
IIGR   41   320    CONTINUE
IIGR   42          BS=0.0
IIGR   43          DO 380 K=1,N
IIGR   44   380    BS=BS+T(K,J)*R(K,J)
IIGR   45   C
IIGR   46          IF (BS.NE.0.0) GO TO 440
IIGR   47          PRINT 1000,J
IIGR   48          DO 400 K=1,N
IIGR   49   400    TT(K)=1.0
IIGR   50          CALL MULT (W,B,TT,N,MB)
IIGR   51          DO 420 K=1,N
IIGR   52          R(K,J)=W(K)
IIGR   53   420    T(K,J)=TT(K)
IIGR   54          GO TO 310
IIGR   55   440    BS=SQRT(BS)
IIGR   56          DO 460 K=1,N
IIGR   57          T(K,J)=T(K,J)/BS
IIGR   58   460    R(K,J)=R(K,J)/BS
IIGR   59   300    CONTINUE
IIGR   60   C
IIGR   61          RETURN
IIGR   62   1000   FORMAT (14HOITERN VECTOR I3,16H HAS ZERO LENGTH  )
IIGR   63          END
```

```
ERRA    1          SUBROUTINE ERRAN (A,B,T,W,TT,MAXA,EIGV,NEIV,BUP,BLO,ERRV,BUPC,
ERRA    2         1N,MA,NWA,NWB,NROOT,NC,SHIFT,TEND,NSCH)
ERRA    3   C
ERRA    4          DIMENSION A(NWA),B(NWB),T(N,1),TT(1),W(1),EIGV(1),BUP(1),BLO(1),
ERRA    5         1ERRV(1),BUPC(1)
ERRA    6          INTEGER MAXA(N),NEIV(1)
ERRA    7          LOGICAL CHECK
ERRA    8   C
ERRA    9          RTOL=1.0E-02
ERRA   10          STOL=1.0E-10
ERRA   11   C
ERRA   12   C     CALCULATE ERROR BOUNDS FOR EIGENVALUES
```

```
ERRA  13          SHIFT=0.0
ERRA  14          CALL BAN (B,A,TT,MAXA,N,NWB,NWA,SHIFT,NSCH,1)
ERRA  15          REWIND 2
ERRA  16          WRITE (2) B,MAXA
ERRA  17          DO 100 L=1,NC
ERRA  18          ROOT=EIGV(L)
ERRA  19          REWIND 1
ERRA  20          READ (1) A
ERRA  21          READ (1) B
ERRA  22          DO 110 I=1,NWB
ERRA  23  110     A(I)=A(I)-ROOT*B(I)
ERRA  24          DO 120 I=1,N
ERRA  25  120     TT(I)=T(I,L)
ERRA  26          CALL MULT (W,A,TT,N,MA)
ERRA  27          DO 130 I=1,N
ERRA  28  130     TT(I)=W(I)
ERRA  29          REWIND 2
ERRA  30          READ (2) B,MAXA
ERRA  31          CALL BAN (B,A,TT,MAXA,N,NWB,NWA,SHIFT,NSCH,2)
ERRA  32          ERR=0.0
ERRA  33          DO 140 I=1,N
ERRA  34  140     ERR=ERR+TT(I)*W(I)
ERRA  35          ERRV(L)=SQRT(ERR)
ERRA  36          BUP(L)=EIGV(L)+ERRV(L)
ERRA  37  100     BLO(L)=EIGV(L)-ERRV(L)
ERRA  38          DO 160 I=1,NC
ERRA  39  160     ERRV(I)=ERRV(I)/EIGV(I)
ERRA  40          PRINT 1000
ERRA  41          PRINT 1005,(BUP(I),I=1,NC)
ERRA  42          PRINT 1005,(BLO(I),I=1,NC)
ERRA  43          PRINT 1005,(ERRV(I),I=1,NC)
ERRA  44          DO 170 I=1,NC
ERRA  45          BUP(I)=BUP(I)*(1.0+STOL)
ERRA  46  170     BLO(I)=BLO(I)*(1.0-STOL)
ERRA  47  C
ERRA  48  C     REDEFINE NROOT IF NECESSARY
ERRA  49  200     IF (ERRV(NROOT).LE.FTOL) GO TO 220
ERRA  50          NROOT=NROOT-1
ERRA  51          IF (NROOT.GE.1) GO TO 200
ERRA  52          PRINT 1010
ERRA  53          RETURN
ERRA  54  C
ERRA  55  C     FIND UPPER BOUNDS ON EIGENVALUE CLUSTERS
ERRA  56  220     DO 240 I=1,NROOT
ERRA  57  240     NEIV(I)=1
ERRA  58          IF (NROOT.NE.1) GO TO 260
ERRA  59          BUPC(1)=BUP(1)
ERRA  60          LM=1
ERRA  61          I=2
ERRA  62          GO TO 295
ERRA  63  260     L=1
ERRA  64          I=2
ERRA  65  270     IF (BUP(I-1).LE.BLO(I)) GO TO 280
ERRA  66          NEIV(L)=NEIV(L)+1
ERRA  67          I=I+1
ERRA  68          IF (I.LE.NROOT) GO TO 270
ERRA  69  280     BUPC(L)=BUP(I-1)
ERRA  70          IF (I.GT.NROOT) GO TO 290
ERRA  71          L=L+1
ERRA  72          I=I+1
ERRA  73          IF (I.LE.NROOT) GO TO 270
ERRA  74          BUPC(L)=BUP(I-1)
ERRA  75  290     LM=L
ERRA  76  295     IF (BUP(I-1).LE.BLO(I)) GO TO 300
ERRA  77          IF (ERRV(I).GT.FTOL) GO TO 300
ERRA  78          BUPC(L)=BUP(I)
ERRA  79          NEIV(L)=NEIV(L)+1
ERRA  80          NROOT=NROOT+1
ERRA  81          IF (NROOT.EQ.NC) GO TO 300
ERRA  82          I=I+1
ERRA  83          GO TO 295
ERRA  84  C
ERRA  85  C     FIND INTERVAL OF MISSING ROOTS USING EIGENVALUE SEPARATION THM
ERRA  86  300     REWIND 1
ERRA  87          READ (1) A
ERRA  88          READ (1) B
ERRA  89          PRINT 1020
ERRA  90          PRINT 1005,(BUPC(I),I=1,LM)
ERRA  91          PRINT 1030
ERRA  92          PRINT 1006,(NEIV(I),I=1,LM)
ERRA  93          NRO=0
ERRA  94          LL=LM-1
ERRA  95          IF (LM.EQ.1) GO TO 310
ERRA  96  330     DO 320 I=1,LL
ERRA  97  320     NEIV(L)=NEIV(L)+NEIV(I)
ERRA  98          L=L-1
ERRA  99          LL=LL-1
ERRA 100          IF (L.NE.1) GO TO 330
ERRA 101  310     DO 340 I=1,LM
ERRA 102          IF (NEIV(I).GE.NROOT) GO TO 350
ERRA 103  340     CONTINUE
ERRA 104  350     SHIFT=BUPC(I)
ERRA 105          II=NEIV(I)
ERRA 106          PRINT 1040,SHIFT
ERRA 107          CALL BAN (A,B,TT,MAXA,N,NWA,NWB,SHIFT,NSCH,1)
ERRA 108          IF (NSCH.EQ.II) GO TO 360
ERRA 109          IF ((NSCH-II).LT.NRO) GO TO 370
ERRA 110          NRO=NSCH-II
ERRA 111          TEND=SHIFT
ERRA 112          NSCHT=NSCH
ERRA 113  370     I=I-1
ERRA 114          IF (I.NE.0) GO TO 350
ERRA 115          SHIFT=0.0
ERRA 116          NSCH=0
ERRA 117          GO TO 380
ERRA 118  360     PRINT 1050,II
ERRA 119          IF (NRO.NE.0) GO TO 380
ERRA 120          NROOT=0
ERRA 121          GO TO 600
ERRA 122  380     NROOT=NSCHT-NSCH
ERRA 123          PRINT 1060,NRO,SHIFT,TEND
ERRA 124  600     RETURN
ERRA 125  C
ERRA 126  1005    FORMAT (1HC,6E22.14)
ERRA 127  1006    FORMAT (1HC,6I22)
ERRA 128  1010    FORMAT (46HTHERE IS NO SENSE IN CARRYING OUT A SS CHECK  )
ERRA 129  1020    FORMAT (37HUPPER BOUNDS ON EIGENVALUE CLUSTERS  )
ERRA 130  1030    FORMAT (34HNO OF EIGENVALUES IN EACH CLUSTER  )
ERRA 131  1040    FORMAT (26HCHECK APPLIED AT SHIFT = E22.14)
ERRA 132  1000    FORMAT (1H1,68HPRINT OF UPPER AND LOWER BOUNDS ON EIGENVALUES AND
ERRA 133         1 THEIR REL ERRORS  )
ERRA 134  1050    FORMAT (14H0WE FOUND THE I4,25H LOWEST ROOTS AND VECTORS )
ERRA 135  1060    FORMAT (10HOTHERE ARE I4,29H EIGENVALUES MISSING BETWEEN E22.14,
ERRA 136         14H AND E22.14)
ERRA 137          END
```

II.3 Program MODES

Program MODES calculates the smallest eigenvalues and associated vectors in the problem $Av = \lambda Bv$, when matrix A is positive definite and B is diagonal non-negative definite. Matrix A may have any order and bandwidth. It is assumed that A and B are stored in blocks on tapes NSTIF and NMASS, respectively (see Chapter 5).

The program is called with six parameters:

NEQ = order of matrix A

MBAND = half bandwidth of A including diagonal

NBLOCK = number of blocks

NEQB = number of equations per block

NF = number of required eigenvalues and associated eigenvectors

MTOT = total blank common storage area specified. The program stops at the beginning of execution if more storage is needed.

For storage a blank common area COMMON A(MTOT) is specified. The storage used in each subroutine is dynamically allocated from this area. Therefore, MTOT must be the maximum of the storage needed in any one of the subroutines used.

Six tapes are specified in the program.

NSTIF = stores initially matrix A

NMASS = stores matrix B

NRED = stores the reduced matrix

NL,NR,NT = working tapes

Tapes NRED, NL, NR and NT are not used when NBLOCK = 1.

All tolerances have been set for the CDC 6400 with a 48 bit mantissa in floating point arithmetic.

If NBLOCK = 1 program SECANTD is called. This program uses the determinant search technique described in Chapter 3. In this case the total storage required is N*(MBAND + 2*NC + 4) + 5*NC, where NC = NF + NIM. The parameter NIM has been set equal to 3 and allows the calculation of 3 more eigenvalues than required (see Appendix II.1). SECANTD calculates the eigenvalues to about 12 digit precision.

When NBLOCK > 1 program SSPACEB is called to calculate the required eigenvalues and vectors using the subspace iteration c in Table 1. SSPACEB calls the following programs:

DECOMP - factorizes (A - μB) which is read from tape NSTIF, where μ is the shift.

Required storage: NEQB*(2*MBAND + 1)

INVECT - generates the initial transformation vectors.

Required storage: NEQB*(NV+1) + NV, where NV = min (2*NF, NF + 8).

REDBAK - does the vector reductions and back substitutions.

Required storage: NEQB*(MBAND + 2*NV + NV*NTB+1) + 2*NV, where NTB = ((MBAND - 2)/NEQB) + 1.

EIGSOL - calculates the projections of A and B, calls JACOBI to solve the small generalized eigenvalue problem, checks for convergence and calculates either B times the new transformation vectors or the orthonormalized eigen-vectors.

Required storage: NV*(3*NV + 2*NEQB + 3) + NEQB.

SCHECK - finds the shift μ for the Sturm Sequence check and puts $(A - \mu B)$ on tape NSTIF.

Required storage: NEQB*(MBAND + 1) + 6*NV.

The Sturm Sequence check is not carried out if we set the logical variable CHECK in SSPACEB equal to .FALSE.. At convergence, SSPACEB has calculated the eigenvalues to about 5 digit precision. For solution we need at least min {2*NF, NF + 8} nonzero diagonal elements in B. Systems of order smaller than about 100 should be analyzed using NBLOCK = 1.

```
MODE   1          SUBROUTINE MODES (NEQ,MBAND,NBLOCK,NEQB,NF,MTOT)
MODE   2  C
MODE   3  C    PROGRAM TO COMPUTE SMALLEST EIGENVALUES AND ASSOCIATED VECTORS IN
MODE   4  C    THE GENERALIZED EIGENVALUE PROBLEM
MODE   5  C                    A*V=RT*B*V  (A POS DEF,B DIAG NONNEG DEF)
MODE   6  C
MODE   7  C                                  PROGRAMMED  BY  K.J.  BATHE
MODE   8  C                                  SESM  UNIV OF CALIF  BERKELEY  1971
MODE   9  C
MODE  10          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
MODE  11          COMMON A(1)
MODE  12  C
MODE  13  C
MODE  14          NSTIF=4
MODE  15          NMASS=9
MODE  16          NRED=1
MODE  17          NL=2
MODE  18          NR=3
MODE  19          NT=7
MODE  20          PRINT 1000,NEQ,MBAND,NBLOCK,NEQB,NF
MODE  21  C
MODE  22          IF (NBLOCK.GT.1) GO TO 300
MODE  23  C
MODE  24          NTM=3
MODE  25          NC=NF+NTM
MODE  26          NWA=NEQ*MBAND
MODE  27          N2=1+NWA
MODE  28          N3=N2+NEQ
MODE  29          N4=N3+NEQ
MODE  30          N5=N4+NEQ
MODE  31          N6=N5+NEQ
MODE  32          N7=N6+NEQ*NC
MODE  33          N8=N7+NEQ*NC
MODE  34          N9=N8+NC
MODE  35          N10=N9+NC
MODE  36          N11=N10+NC
MODE  37          N12=N11+NC
MODE  38          N13=N12+NC
MODE  39  C
MODE  40          IF (MTOT-N13) 100,200,200
MODE  41   100    PRINT 1010,N13
MODE  42          STOP
MODE  43  C
MODE  44   200    CALL SECANTD (A(1),A(N2),A(N3),A(N4),A(N5),A(N6),A(N7),A(N8),A(N9
MODE  45         1),A(N10),A(N11),A(N12),NEQ,MBAND,NWA,NF,NC)
MODE  46          GO TO 600
MODE  47  C
MODE  48   300    NWA=NEQB*MBAND
MODE  49          NV=2*NF
MODE  50          IF (NF.GT.8) NV=NF+8
MODE  51          NWV=NV*NEQB
MODE  52          NTB=(MBAND-2)/NEQB+1
MODE  53          IF (NTB.GE.NBLOCK) NTB=NBLOCK-1
MODE  54          NWVV=NWV*(NTB+1)
MODE  55          N=2*NWA+NEQB
MODE  56          N2=NWV+NEQB+NV
MODE  57          N3=NWA+NWV+NWVV+NEQB+2*NV
MODE  58          N4=NV*(3*NV+2*NEQB+3)+NEQB
MODE  59          N5=NWA+6*NV+NEQB
MODE  60  C
MODE  61          IF (N2.GT.N) N=N2
MODE  62          IF (N3.GT.N) N=N3
MODE  63          IF (N4.GT.N) N=N4
MODE  64          IF (N5.GT.N) N=N5
MODE  65          IF (MTOT-N) 400,500,500
MODE  66   400    PRINT 1010,N
MODE  67          STOP
MODE  68  C
MODE  69   500    CALL SSPACEB (NEQ,MBAND,NBLOCK,NEQB,NF,NV,NWA,NWV,NWVV,NTB)
MODE  70  C
MODE  71   600    RETURN
MODE  72  C
MODE  73   1000   FORMAT (1H1,20HPROBLEM INFORMATION  //
MODE  74         1          /3CH NO OF EQUATIONS ..............  I4
MODE  75         2          /3CH 1/2 BANDWIDTH OF A ...........  I4
MODE  76         3          /3CH NO OF BLOCKS .................  I4
MODE  77         4          /3CH NO OF EQNS PER BLOCK .........  I4
MODE  78         5          /3CH NO OF FREQUENCIES REQ'D ......  I4  )
MODE  79   1010   FORMAT (40HOFOR EXECUTION NEED TO INCREASE MTOT TO  I6)
MODE  80          END
```

```
SECA   1          SUBROUTINE SECANTD (A,B,V,MAXA,W,VV,WW,ROOT,TIM,ERRVL,ERRVR,
SECA   2         1NITE,N,MA,NWA,NRCDT,NC)
SECA   3  C
SECA   4          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
SECA   5          DIMENSION A(NWA),B(N),V(1),VS(1),W(1),VV(N,1),WW(N,1),ROOT(1),
SECA   6         1TIM(1),ERRVL(1),ERRVR(1)
SECA   7          INTEGER NITE(1),MAXA(1)
SECA   8  C
SECA   9  C    FOLLOWING TOLERANCES ARE SET FOR THE CDC 6400
SECA  10          ACTOL=1.0E-04
SECA  11          RCBTOL=1.0E-06
SECA  12          RTOL=1.0E-10
SECA  13          RQTOL=1.0E-12
SECA  14          SCALE=2.0**900
SECA  15  C
SECA  16          NTF=5
SECA  17          IITEM=10
SECA  18          NITEM=40
SECA  19  C
SECA  20          REWIND NMASS
SECA  21          READ (NMASS) B
SECA  22  C
SECA  23          ETA=2.0
SECA  24          NCV=0
SECA  25          JR=1
SECA  26          NSK=0
SECA  27          NWA=N*MA
SECA  28          ISC=1000
SECA  29  C
SECA  30          CALL SECOND (TIM1)
SECA  31          RA=0.0
SECA  32          RR=0.0
SECA  33          CALL BANDET (A,B,V,MAXA,N,NWA,RA,NSCH,DETA,ISC,1)
SECA  34          FA=DETA
SECA  35          FR=FA
SECA  36          DETR=DETA
SECA  37  C
SECA  38  C    FIND LOWER BOUND ON SMALLEST EIGENVALUE
SECA  39          PRINT 1010
SECA  40          DO 100 I=1,N
SECA  41   100    W(I)=B(I)
SECA  42          RT=0.0
SECA  43          IITE=0
SECA  44          KK=3
SECA  45   110    IITE=IITE+1
SECA  46          DO 120 I=1,N
SECA  47   120    V(I)=W(I)
SECA  48          CALL BANDET (A,B,V,MAXA,N,NWA,RA,NSCH,DETA,ISC,KK)
SECA  49          KK=2
SECA  50          RQT=0.0
SECA  51          DO 130 I=1,N
SECA  52   130    RQT=RQT+W(I)*V(I)
SECA  53          DO 180 I=1,N
SECA  54   180    W(I)=R(I)*V(I)
SECA  55          RQB=0.0
SECA  56          DO 140 I=1,N
SECA  57   140    RQB=RQB+W(I)*V(I)
SECA  58          RQ=RQT/RQB
SECA  59          PRINT 1004,RQ
SECA  60          RS=SQRT(RQB)
SECA  61          TOL=ABS(RQ-RT)/RQ
SECA  62          IF (TOL.LT.RCBTOL) GO TO 150
SECA  63          DO 160 I=1,N
```

```
SECA   64   160   W(I)=W(I)/BS
SECA   65         RT=RQ
SECA   66         IF (IITF.LT.IITEM) GO TO 110
SECA   67  C
SECA   68   150   DO 170 I=1,N
SECA   69   170   V(I)=V(I)/BS
SECA   70         RB=RQ*(1.0-AMIN1(0.1,100*TOL))
SECA   71         IS=0
SECA   72   230   CALL BANDET (A,B,V,MAXA,N,NWA,RB,NSCH,DETB,ISC,1)
SECA   73         PRINT 1020,RB,NSCH
SECA   74         FB=DETB
SECA   75         IF (NSCH.EQ.0) GO TO 300
SECA   76         IS=IS+1
SECA   77         IF (IS.LE.NTF) GO TO 240
SECA   78         PRINT 1030
SECA   79         STOP
SECA   80   240   RB=RB/(NSCH+1)
SECA   81         GO TO 230
SECA   82  C
SECA   83  C   ITERN FOR INDIVIDUAL ROOTS
SECA   84   300   PRINT 1040
SECA   85         NITE(JR)=1
SECA   86         PRINT 1050,JR,NITE(JR),RA,DETA,FA,ETA,ISC
SECA   87         NITE(JR)=2
SECA   88         PRINT 1050,JR,NITE(JR),RB,DETB,FB,ETA,ISC
SECA   89  C
SECA   90  C   WE STOP WHEN WE HAVE THE REQUIRED NO OF ROOTS SMALLER THAN RC AND
SECA   91  C   NOV=0
SECA   92   310   IF (NSCH.GE.NROOT) GO TO 900
SECA   93  C
SECA   94         DIF=FB-FA
SECA   95         IF (DIF.NE.0.0) GO TO 320
SECA   96         PRINT 1060
SECA   97         GO TO 900
SECA   98   320   DEL=FB*(RB-RA)/DIF
SECA   99         RC=RB-ETA*DEL
SECA  100         TOL=RCBTOL*RC
SECA  101         IF (ABS(RC-RB).GT.TOL) GO TO 330
SECA  102         PRINT 1070
SECA  103         ROOT(JR)=RB
SECA  104         GO TO 400
SECA  105  C
SECA  106   330   CALL BANDET (A,B,V,MAXA,N,NWA,RC,NSCH,DETC,ISC,1)
SECA  107         FC=DETC
SECA  108         NITE(JR)=NITE(JR)+1
SECA  109         IF (JR.EQ.1) GO TO 340
SECA  110         JJ=JR-1
SECA  111         DO 350 K=1,JJ
SECA  112   350   FC=FC/(RC-ROOT(K))
SECA  113   340   PRINT 1050,JR,NITE(JR),RC,DETC,FC,ETA,ISC
SECA  114  C
SECA  115  C   IF WE HAVE MORE SIGNCHANGES THAN EIGENVALUES SMALLER THAN RC WE
SECA  116  C   START INV. ITERATION
SECA  117         NES=0
SECA  118         IF (JR.EQ.1) GO TO 380
SECA  119         DO 360 I=1,JJ
SECA  120   360   IF (ROOT(I).LT.RC) NES=NES+1
SECA  121   380   NOV=NSCH-NES
SECA  122         IF (NOV.EQ.0) GO TO 370
SECA  123         PRINT 1080,NOV
SECA  124         RCOT(JR)=RC
SECA  125         IF (NOV.GT.1) NSK=1
SECA  126  C
SECA  127         GO TO 400
SECA  128   370   RB=RA
SECA  129         FB=FA
SECA  130         DETB=DETA
SECA  131         RA=RB
SECA  132         FA=FB
SECA  133         DETA=DETB
SECA  134         RB=RC
SECA  135         FB=FC
SECA  136         DETB=DETC
SECA  137  C
SECA  138  C   WE RESET ETA IF NECESSARY
SECA  139         TOL=RB*ACTOL
SECA  140         IF (ABS(RA-RB).LT.TOL) ETA=ETA*2
SECA  141         IF (NITE(JR).LE.NITEM) GO TO 310
SECA  142         PRINT 1015,NITE(JR),JR
SECA  143         GO TO 900
SECA  144  C
SECA  145  C   CHECK FOR STORAGE
SECA  146   400   IF (JR.LE.NC) GO TO 405
SECA  147         PRINT 1090
SECA  148         GO TO 900
SECA  149  C
SECA  150   405   NOR=JR-1
SECA  151         CALL SECOND (TIM3)
SECA  152         PRINT 1100,NOR
SECA  153         IF (JR.EQ.1) GO TO 410
SECA  154         DO 420 I=1,N
SECA  155   420   V(I)=1.0
SECA  156         KK=3
SECA  157   410   DO 430 I=1,N
SECA  158   430   W(I)=B(I)*V(I)
SECA  159         IS=0
SECA  160         RTA=0.0
SECA  161         GO TO 510
SECA  162  C
SECA  163  C   INVERSE ITERN
SECA  164   440   NITE(JR)=NITE(JR)+1
SECA  165         DO 450 I=1,N
SECA  166   450   V(I)=W(I)
SECA  167         CALL BANDET (A,B,V,MAXA,N,NWA,RC,NSCH,DETC,ISC,KK)
SECA  168         IF (IS.EQ.1) GO TO 460
SECA  169         KK=2
SECA  170         RQT=0.0
SECA  171         DO 470 I=1,N
SECA  172   470   RQT=RQT+W(I)*V(I)
SECA  173         DO 475 I=1,N
SECA  174   475   W(I)=B(I)*V(I)
SECA  175         RQB=0.0
SECA  176         DO 480 I=1,N
SECA  177   480   RQB=RQB+W(I)*V(I)
SECA  178         RQ=RQT/RQB
SECA  179         RT=ROOT(JR)+RQ
SECA  180         PRINT 1110,JR,NITE(JR),RT,RQ
SECA  181         TOL=RT*RQTOL
SECA  182         IF (ABS(RT-RTA).GT.TOL) GO TO 510
SECA  183         IS=1
SECA  184         GO TO 440
SECA  185  C
SECA  186   510   RTA=RT
SECA  187         BS=SQRT(RQB)
SECA  188         DO 490 I=1,N
SECA  189   490   W(I)=W(I)/BS
SECA  190         IF (NOR.EQ.0) GO TO 550
SECA  191         DO 520 K=1,NOR
SECA  192         AL=0.0
SECA  193         DO 530 I=1,N
SECA  194   530   AL=AL+VV(I,K)*W(I)
SECA  195         DO 540 I=1,N
SECA  196   540   W(I)=W(I)-AL*WW(I,K)
SECA  197   520   CONTINUE
SECA  198  C
SECA  199   550   IF (NITE(JR).LE.NITEM) GO TO 440
SECA  200         PRINT 1015,NITE(JR),JR
SECA  201         GO TO 900
SECA  202  C
SECA  203   460   RQT=0.0
SECA  204         ERRT=RQB
SECA  205         DO 570 I=1,N
SECA  206   570   RQT=RQT+V(I)*W(I)
SECA  207         DO 560 I=1,N
SECA  208   560   W(I)=B(I)*V(I)
SECA  209         RQB=0.0
SECA  210         DO 580 I=1,N
SECA  211   580   RQB=RQB+V(I)*W(I)
SECA  212  C
SECA  213  C   OBTAIN A RATHER LARGE ERROR BOUND
```

```
SECA 214          RQ=SQT/RQB
SECA 215          RCOT(JR)=RCOT(JR)+RQ
SECA 216          ERB=SQRT(ERRT/RQB)
SECA 217          ERRVL(JR)=ROOT(JR)-ERR
SECA 218          ERRVR(JR)=ROOT(JR)+ERR
SECA 219  C
SECA 220          BS=SQRT(PQB)
SECA 221          DO 590 I=1,N
SECA 222          W(I)=W(I)/BS
SECA 223   590     V(I)=V(I)/BS
SECA 224          DO 600 I=1,N
SECA 225          WW(I,JR)=W(I)
SECA 226   600     VV(I,JR)=V(I)
SECA 227  C
SECA 228          CALL SECOND (TIM2)
SECA 229          TIM3=TIM2-TIM3
SECA 230          PRINT 1120,TIM3
SECA 231          TIM(JR)=TIM2-TIM1
SECA 232          TIM1=TIM2
SECA 233  C
SECA 234  C    DECIDE STRATEGY FOR ITERN TOWARDS NEXT ROOT
SECA 235          TOL=RTOL*ROOT(JR)
SECA 236          IF (NOV.GT.0) GO TO 700
SECA 237          IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 710
SECA 238          IF (RA.GT.0.0) GO TO 720
SECA 239          RA=RB/2.
SECA 240          CALL BANDET (A,B,V,MAXA,N,NWA,RA,NSCH,DETA,ISC,1)
SECA 241          FA=DETA
SECA 242   720     RB=RA
SECA 243          FB=FA
SECA 244          DETB=DETA
SECA 245          RA=RR
SECA 246          FA=FR
SECA 247          DETA=DETR
SECA 248          GO TO 710
SECA 249  C
SECA 250   700     IF (ROOT(JR).GT.RC) NSK=1
SECA 251          IF (NSK.EQ.1) GO TO 730
SECA 252          IF (ABS(RC-ROOT(JR)).LT.TOL) GO TO 740
SECA 253          IF (ABS(ROOT(JR)-RB).LT.TOL) GO TO 750
SECA 254          RA=RB
SECA 255          FA=FB
SECA 256          DETA=DETB
SECA 257   750     RB=RC
SECA 258          FB=FC
SECA 259          DETB=DETC
SECA 260          GO TO 710
SECA 261   740     IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 710
SECA 262          IF (RA.GT.0.0) GO TO 760
SECA 263          RA=RB/2.
SECA 264          CALL BANDET (A,B,V,MAXA,N,NWA,RA,NSCH,DETA,ISC,1)
SECA 265          FA=DETA
SECA 266   760     RB=RA
SECA 267          FB=FA
SECA 268          DETB=DETA
SECA 269          RA=RR
SECA 270          FA=FR
SECA 271          DETA=DETR
SECA 272   710     FA=FA/(RA-ROOT(JR))
SECA 273          FB=FB/(RB-ROOT(JR))
SECA 274          JR=JR+1
SECA 275          ETA=2.0
SECA 276          GO TO 300
SECA 277  C
SECA 278   730     IF (RA.GT.0.0) GO TO 780
SECA 279          RA=RB/2.
SECA 280          CALL BANDET (A,B,V,MAXA,N,NWA,RA,NSCH,DETA,ISC,1)
SECA 281          FA=DETA
SECA 282   780     IF (ABS(ROOT(JR)-RB).GT.TOL) GO TO 770
SECA 283          RR=RA
SECA 284          FR=FA
SECA 285          DETR=DETA
SECA 286          RA=RB
SECA 287          FA=FB
SECA 288          DETA=DETB
SECA 289   770     FA=FA/(RA-ROOT(JR))
SECA 290          FB=FB/(RB-ROOT(JR))
SECA 291          FR=FR/(RR-ROOT(JR))
SECA 292          IF (ROOT(JR).LT.RC) NOV=NOV-1
SECA 293          JR=JR+1
SECA 294          NITE(JR)=0
SECA 295          RCOT(JR)=RC
SECA 296          IF (NOV.GT.0) GO TO 400
SECA 297          NSK=0
SECA 298          ETA=2.0
SECA 299          GO TO 300
SECA 300  C
SECA 301   900     NROOT=JR-1
SECA 302          IF (NROOT.EQ.0) RETURN
SECA 303          PRINT 1130
SECA 304          PRINT 1004,(ROOT(J),J=1,NROOT)
SECA 305          PRINT 1140
SECA 306          PRINT 1006,(NITE(J),J=1,NROOT)
SECA 307          PRINT 1150
SECA 308          PRINT 1008,(TIM(J),J=1,NROOT)
SECA 309          PRINT 1160
SECA 310          PRINT 1004,(ERRVL(J),J=1,NROOT)
SECA 311          PRINT 1004,(ERRVR(J),J=1,NROOT)
SECA 312  C
SECA 313  C   ARRANGE EIGENVALUES AND VECTORS IN ASCENDING ORDER
SECA 314          IF (JR.EQ.2) GO TO 950
SECA 315          JR=JR-2
SECA 316   910     IS=0
SECA 317          DO 920 I=1,JR
SECA 318          IF (ROOT(I+1).GE.ROOT(I)) GO TO 920
SECA 319          IS=IS+1
SECA 320          RT=ROOT(I+1)
SECA 321          ROOT(I+1)=ROOT(I)
SECA 322          ROOT(I)=RT
SECA 323          DO 930 K=1,N
SECA 324          RT=VV(K,I+1)
SECA 325          VV(K,I+1)=VV(K,I)
SECA 326   930     VV(K,I)=RT
SECA 327   920     CONTINUE
SECA 328          IF (IS.GT.0) GO TO 910
SECA 329  C
SECA 330   950     PRINT 1170
SECA 331          NROOT=NSCH
SECA 332          PRINT 1130
SECA 333          PRINT 1004,(ROOT(J),J=1,NROOT)
SECA 334          PRINT 1180
SECA 335          DO 960 J=1,NROOT
SECA 336   960     PRINT 1002,(VV(I,J),I=1,N)
SECA 337          RETURN
SECA 338  C
SECA 339   1002   FORMAT (1H ,12E11.4)
SECA 340   1004   FORMAT (1HO,6E20.12)
SECA 341   1006   FORMAT (1HO,6I20)
SECA 342   1008   FORMAT (1HO,6F20.2)
SECA 343   1010   FORMAT (1H1,63HINVERSE ITERN GIVES FOLLOWING APPROXIMATN TO LOWES
     1T EIGENVALUE  )
SECA 345   1015   FORMAT (41HOWE ABANDON ITERN BECAUSE NO OF ITERN IS I3,9H FOR ROO
     1T I3 )
SECA 347   1020   FORMAT (5HORB = E20.12,7H NSCH = I4)
SECA 348   1030   FORMAT (30HOWE BETTER CHECK THE MATRICES  )
SECA 349   1040   FORMAT (1H1,4X,4HROOT,4X,4HNITE,18X,2HRC,15X,12HDET (A-RC*B),15X,
     /2HFC,13X,3HETA,4X,3HISC)
SECA 351   1050   FORMAT (1HC,4X,I4,4X,I4,8X,3E12.4,F7.2,I6)
SECA 352   1060   FORMAT (42HOTHE DEFLATED POLYNOMIAL HAS NO MORE ROOTS  )
SECA 353   1070   FORMAT (29HO(RC-RB) IS SMALLER THAN TOL )
SECA 354   1080   FORMAT (16HOWE JUMPED OVER I4,16H UNKNOWN ROOT(S)  )
SECA 355   1090   FORMAT (1H1,36H  NO MORE STORAGE FOR VECTORS  WE QUIT  )
SECA 356   1100   FORMAT (1HC,34X,4HROOT,18X,2HRQ,18X,4HNOR=,I2)
SECA 357   1110   FORMAT (1HO,4X,I4,4X,I4,8X,2E22.14)
SECA 358   1120   FORMAT (20HOTIME FOR INV ITERN F5.2)
SECA 359   1130   FORMAT (20HOTHE EIGENVALUES ARE  /)
SECA 360   1140   FORMAT (42HONO OF ITERATIONS FOR EACH EIGENVALUE ARE   /)
SECA 361   1150   FORMAT (30HOTIME USED FOR EACH EIGENVALUE /)
SECA 362   1160   FORMAT (43HOFOLLOWING ARE ERROR BOUNDS ON EIGENVALUES )
SECA 363   1170   FORMAT (1H1,62HWE ACCEPT FOLLOWING EIGENVALUES AND VECTORS ARRANG
```

```
SECA 364        1ED IN ORDER )
SECA 365    1180  FORMAT (22HTHE EIGENVECTORS ARE  //)
SECA 366    C
SECA 367          END
```

```
BAND   1          SUBROUTINE BANDET (A,B,V,MAXA,NN,NWA,RA,NSCH,DET,ISCALE,KK)
BAND   2    C
BAND   3          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
BAND   4          DIMENSION A(NWA),B(1),V(1),MAXA(1)
BAND   5    C
BAND   6          NR=NN-1
BAND   7          IF (KK-2) 100,700,800
BAND   8    C
BAND   9     100  TOL=1.0E+07
BAND  10          RTOL=1.0E-10
BAND  11          SCALE=2.0**900
BAND  12          NTF=3
BAND  13          IS=1
BAND  14     120  REWIND NSTIF
BAND  15          READ (NSTIF) A
BAND  16          DO 140 I=1,NN
BAND  17     140  A(I)=A(I)-RA*B(I)
BAND  18     160  IF (NWA.EQ.NN) GO TO 230
BAND  19          DO 200 N=1,NR
BAND  20          IH=N+NWA-NN
BAND  21     210  IF (A(IH)) 220,215,220
BAND  22     215  IH=IH-NN
BAND  23          GO TO 210
BAND  24     220  MAXA(N)=IH
BAND  25          PIV=A(N)
BAND  26          IF (PIV) 221,226,221
BAND  27     221  IL=N+NN
BAND  28          L=N
BAND  29          DO 240 I=IL,IH,NN
BAND  30          L=L+1
BAND  31          C=A(I)
BAND  32          IF (C) 225,240,225
BAND  33     225  C=C/PIV
BAND  34          IF (ABS(C).LT.TOL) GO TO 235
BAND  35     226  IS=IS+1
BAND  36          IF (IS.LE.NTF) GO TO 245
BAND  37          PRINT 1000,NTF
BAND  38          STOP
BAND  39     245  RA=RA*(1.0-RTOL)
BAND  40          GO TO 120
BAND  41     235  J=L-I
BAND  42          DO 260 K=I,IH,NN
BAND  43     260  A(K+J)=A(K+J)-C*A(K)
BAND  44          A(I)=C
BAND  45     240  CONTINUE
BAND  46     200  CONTINUE
BAND  47     230  IF (A(NN).NE.0.0) GO TO 280
BAND  48          AA=ABS(A(1))
BAND  49          DO 290 I=2,NR
BAND  50     290  AA=AA+ABS(A(I))
BAND  51          A(NN)=-(AA/NR)*1.0E-16
BAND  52    C
BAND  53     280  NSCH=0
BAND  54          ISC=0
BAND  55          DET=1.0
BAND  56          DO 300 I=1,NN
BAND  57          IF (ABS(DET).LT.SCALE) GO TO 320
BAND  58          DET=DET/SCALE
BAND  59          ISC=ISC+1
BAND  60     320  DET=DET*A(I)
BAND  61     300  IF (A(I).LT.0.) NSCH=NSCH+1
BAND  62    C
BAND  63          IF (ISCALE.LT.1000) GO TO 340
BAND  64          ISCALE=ISC
```

```
BAND  65          GO TO 900
BAND  66     340  IF (ISC-ISCALE) 350,900,370
BAND  67     350  DET=DET/SCALE
BAND  68          GO TO 900
BAND  69     370  DET=DET*SCALE
BAND  70          GO TO 900
BAND  71    C
BAND  72     700  IL=NN
BAND  73          DO 400 N=1,NR
BAND  74          C=V(N)
BAND  75          V(N)=C/A(N)
BAND  76          IF (NWA-NN) 410,400,410
BAND  77     410  IL=IL+1
BAND  78          IH=MAXA(N)
BAND  79          K=N
BAND  80          DO 420 I=IL,IH,NN
BAND  81          K=K+1
BAND  82     420  V(K)=V(K)-C*A(I)
BAND  83     400  CONTINUE
BAND  84          V(NN)=V(NN)/A(NN)
BAND  85    C
BAND  86     800  IF (NWA-NN) 430,900,430
BAND  87     430  N=NN
BAND  88          DO 440 L=2,NN
BAND  89          N=N-1
BAND  90          IL=N+NN
BAND  91          IH=MAXA(N)
BAND  92          K=N
BAND  93          DO 460 I=IL,IH,NN
BAND  94          K=K+1
BAND  95     460  V(N)=V(N)-A(I)*V(K)
BAND  96     440  CONTINUE
BAND  97     900  RETURN
BAND  98    1000  FORMAT (1H1,20H TRIANG FACTORIZATN I3,32H TIMES ABANDONED,CHECK
BAND  99         1MATRICES  )
BAND 100          END
```

```
SSPA   1          SUBROUTINE SSPACE8 (NEQ,MBAND,NBLOCK,NEQB,NF,NV,NWA,NWV,NWVV,NTB)
SSPA   2    C
SSPA   3          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
SSPA   4          COMMON A(1)
SSPA   5          LOGICAL CHECK
SSPA   6    C
SSPA   7          NITEM=12
SSPA   8          CHECK=.TRUE.
SSPA   9    C
SSPA  10    C   FACTORIZE STIFFNESS MATRIX
SSPA  11          N2=1+NWA
SSPA  12          N3=N2+NWA
SSPA  13          CALL SECOND (TIM1)
SSPA  14          CALL DECOMP (A(1),A(N2),A(N3),NEQB,MBAND,NBLOCK,NWA,NTB,NSCH,NEQ)
SSPA  15          CALL SECOND (TIM2)
SSPA  16    C
SSPA  17    C   ESTABLISH STARTING TRANSFORMATION VECTORS ON TAPE NR
SSPA  18          N2=1+NWV
SSPA  19          N3=N2+NEQB
SSPA  20          CALL INVECT (A(1),A(N2),A(N3),NBLOCK,NEQB,NV)
SSPA  21          CALL SECOND (TIM3)
SSPA  22          TIM1=TIM2-TIM1
SSPA  23          TIM2=TIM3-TIM2
SSPA  24          PRINT 1000,TIM1
SSPA  25          PRINT 1010,TIM2
SSPA  26    C
SSPA  27    C   PERFORM SUBSPACE ITERN
SSPA  28          DO 100 I=1,NV
SSPA  29     100  A(I)=0.0
SSPA  30          NITE=0
SSPA  31     200  NITE=NITE+1
SSPA  32          PRINT 1020,NITE
```

```
SSPA  33          CALL SECOND (TIM1)
SSPA  34          N1=1+2*NV
SSPA  35          N2=N1+NWA
SSPA  36          N3=N2+NWV
SSPA  37          N4=N3+NWVV
SSPA  38          CALL REDBAK (A(N1),A(N2),A(N3),A(N4),NEQ0,NV,NWA,NWV,NWVV,NTB,
SSPA  39         1NBLOCK)
SSPA  40  C
SSPA  41  C    SOLVE SUBSPACE EIGENVALUE PROBLEM
SSPA  42          N2=1+NV
SSPA  43          N3=N2+NV
SSPA  44          N4=N3+NV*NV
SSPA  45          N5=N4+NV*NV
SSPA  46          N6=N5+NV*NV
SSPA  47          N7=N6+NWV
SSPA  48          N8=N7+NWV
SSPA  49          N9=N8+NV
SSPA  50          CALL SECOND (TIM2)
SSPA  51          CALL EIGSOL (A(1),A(N2),A(N3),A(N4),A(N5),A(N6),A(N7),A(N8),A(N9)
SSPA  52         1,NF,NV,NBLOCK,NEQB,NITE)
SSPA  53          CALL SECOND (TIM3)
SSPA  54          TIM1=TIM3-TIM1
SSPA  55          TIM2=TIM3-TIM2
SSPA  56          PRINT 1030,TIM1
SSPA  57          PRINT 1040,TIM2
SSPA  58  C
SSPA  59          IF (NITE.LT.NITEM) GO TO 200
SSPA  60  C
SSPA  61          PRINT 1050
SSPA  62          PRINT 1060,(A(I),I=1,NF)
SSPA  63          PRINT 1070
SSPA  64          N1=2*NV+1
SSPA  65          N2=N1+NF*NEQB
SSPA  66          DO 300 L=1,NBLOCK
SSPA  67          PRINT 1075,L
SSPA  68          BACKSPACE NR
SSPA  69          READ (NR) (A(I),I=N1,N2)
SSPA  70          BACKSPACE NR
SSPA  71          DO 320 K=1,NF
SSPA  72          N3=(K-1)*NEQB+2*NV+1
SSPA  73          N4=K*NEQB+2*NV
SSPA  74   320     PRINT 1080,(A(I),I=N3,N4)
SSPA  75   300     CONTINUE
SSPA  76  C
SSPA  77          IF (.NOT.CHECK) GO TO 600
SSPA  78  C
SSPA  79  C    APPLY STURM SEQUENCE CHECK
SSPA  80          CALL SECOND (TIM1)
SSPA  81          N2=1+NV
SSPA  82          N3=N2+NV
SSPA  83          N4=N3+NWA
SSPA  84          N5=N4+NEQB
SSPA  85          N6=N5+NV
SSPA  86          N7=N6+NV
SSPA  87          N8=N7+NV
SSPA  88          CALL SCHECK (A(1),A(N2),A(N3),A(N4),A(N5),A(N6),A(N7),A(N8),NWA,
SSPA  89         1NEQB,NBLOCK,NF,NV,SHIFT,NEI)
SSPA  90          PRINT 1085,SHIFT
SSPA  91          N2=1+NWA
SSPA  92          N3=N2+NWA
SSPA  93          CALL DECOMP (A(1),A(N2),A(N3),NEQB,MBAND,NBLOCK,NWA,NTB,NSCH,NEQ)
SSPA  94          IF (NSCH.EQ.NEI) GO TO 500
SSPA  95          NSCH=NSCH-NEI
SSPA  96          PRINT 1090,NSCH
SSPA  97          GO TO 540
SSPA  98   500     PRINT 1100,NSCH
SSPA  99   540     CALL SECOND (TIM2)
SSPA 100          TIM2=TIM2-TIM1
SSPA 101          PRINT 1110,TIM2
SSPA 102  C
SSPA 103   600     RETURN
SSPA 104  C
SSPA 105   1000    FORMAT (34HOTIME FOR STIFFNESS FACTORIZATION  F6.2)
SSPA 106   1010    FORMAT (42HOTIME FOR GENERATION OF INITIAL TRY-VECTORS  F6.2)
SSPA 107   1020    FORMAT (1H1,11HNO OF ITERN  I4)
SSPA 108   1030    FORMAT (24HOTIME USED IN ITERN STEP  F6.2)
SSPA 109   1040    FORMAT (25HOTIME FOR EIGENVALUE SOLN  F6.2)
SSPA 110   1050    FORMAT (1H-1,26HTHE FINAL EIGENVALUES ARE /)
SSPA 111   1060    FORMAT (1HO,6E22.14)
SSPA 112   1070    FORMAT (40HOPRINT OF EIGENVECTORS  BLOCK BY BLOCK  )
SSPA 113   1075    FORMAT (6HOBLOCK I4)
SSPA 114   1080    FORMAT (1H ,12E11.4)
SSPA 115   1085    FORMAT (1H1,22HCHECK APPLIED AT SHIFT  E22.14)
SSPA 116   1090    FORMAT (10HOTHERE ARE I4,21H EIGENVALUES MISSING  )
SSPA 117   1100    FORMAT (20HOWE FOUND THE LOWEST I4,12H EIGENVALUES  )
SSPA 118   1110    FORMAT (30HOTIME FOR STURM SEQUENCE CHECK  F6.2)
SSPA 119          END
```

```
DECO   1          SUBROUTINE DECOMP (A,B,MAXB,NEQB,MBAND,NBLOCK,NWA,NTB,NSCH,NEQ)
DECO   2  C
DECO   3          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
DECO   4          DIMENSION A(NWA),B(NWA),MAXB(NEQB)
DECO   5  C
DECO   6          NEQB1=NEQB-1
DECO   7          N1=NL
DECO   8          N2=NR
DECO   9          REWIND NSTIF
DECO  10          REWIND NRED
DECO  11          REWIND N1
DECO  12          REWIND N2
DECO  13          NSCH=0
DECO  14  C
DECO  15          DO 600 N=1,NBLOCK
DECO  16          IF (N.NE.1) GO TO 10
DECO  17          READ (NSTIF) A
DECO  18          GO TO 110
DECO  19    10    IF (NTB.EQ.1) GO TO 110
DECO  20          REWIND N1
DECO  21          REWIND N2
DECO  22          READ (N1) A
DECO  23  C
DECO  24  C    FACTORIZE LEADING BLOCK
DECO  25    110   DO 300 I=1,NEQB1
DECO  26          PIV=A(I)
DECO  27          IF (PIV) 120,115,130
DECO  28    115   II=(N-1)*NEQB+I
DECO  29          IF (II.GT.NEQ) GO TO 520
DECO  30          PRINT 1000,II
DECO  31          STOP
DECO  32    120   NSCH=NSCH+1
DECO  33    130   IH=I+NWA-NEQB
DECO  34    140   IF (A(IH)) 160,150,160
DECO  35    150   IH=IH-NEQB
DECO  36          GO TO 140
DECO  37    160   MAXB(I)=IH
DECO  38          JL=I+1
DECO  39          II=I
DECO  40          DO 200 J=JL,NEQB
DECO  41          II=II+NEQB
DECO  42          IF (II-NWA) 170,170,300
DECO  43    170   C=A(II)
DECO  44          IF (C) 180,200,180
DECO  45    180   C=C/PIV
DECO  46          KK=J
DECO  47          MAX=MAXB(I)
DECO  48          DO 250 JJ=II,MAX,NEQB
DECO  49          A(KK)=A(KK)-C*A(JJ)
DECO  50    250   KK=KK+NEQB
DECO  51    200   A(II)=C
DECO  52    300   CONTINUE
DECO  53          IF (A(NEQB)) 80,60,70
DECO  54    60    II=N*NEQB
DECO  55          IF (II.GT.NEQ) GO TO 520
DECO  56          PRINT 1000,II
```

```
DECO  57          STOP
DECO  58    80    NSCH=NSCH+1
DECO  59    70    DO 50 J=NEQB,NWA,NEQB
DECO  60    50    IF (A(J).NE.0.0) MAXB(NEQB)=J
DECO  61    C
DECO  62    C   CARRY OVER INTO TRAILING BLOCKS
DECO  63          DO 400 NN=1,NTB
DECO  64          IF ((NN+N).GT.NBLOCK) GO TO 400
DECO  65          NI=N1
DECO  66          IF ((N.EQ.1).OR.(NN.EQ.NTB)) NI=NSTIF
DECO  67          READ (NI) B
DECO  68          IL=1+NN*NEQB*NEQB
DECO  69          DO 420 I=1,NEQB
DECO  70          II=IL
DECO  71          DO 440 K=1,NEQB
DECO  72          IF (II-NWA) 410,410,440
DECO  73    410   C=A(II)
DECO  74          IF (C) 430,440,430
DECO  75    430   C=C/A(K)
DECO  76          MAX=MAXB(K)
DECO  77          KK=I
DECO  78          DO 460 JJ=II,MAX,NEQB
DECO  79          B(KK)=B(KK)-C*A(JJ)
DECO  80    460   KK=KK+NEQB
DECO  81          A(II)=C
DECO  82    440   II=II-NEQB
DECO  83    420   IL=IL+NEQB
DECO  84          IF (NTB.NE.1) GO TO 480
DECO  85          WRITE (NRED) A,MAXB
DECO  86          DO 500 I=1,NWA
DECO  87    500   A(I)=B(I)
DECO  88          GO TO 600
DECO  89    480   WRITE (N2) B
DECO  90    400   CONTINUE
DECO  91          M=N1
DECO  92          N1=N2
DECO  93          N2=M
DECO  94    520   WRITE (NRED) A,MAXB
DECO  95    600   CONTINUE
DECO  96    C
DECO  97          RETURN
DECO  98    1000  FORMAT (22HOPIVOT IS ZERO IN ROW  I4)
DECO  99          END
```

```
INVE   1          SUBROUTINE INVECT (VA,XM,IEQ,NBLOCK,NEQB,NV)
INVE   2    C
INVE   3          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
INVE   4          DIMENSION VA(NEQB,NV),XM(NEQB),IEQ(1)
INVE   5    C
INVE   6          NV1=NV-1
INVE   7          KK=1
INVE   8          IND=0
INVE   9     90   NBV=KK*((NV1-1)/NBLOCK+1)
INVE  10          IF (NBV.GT.NEQB) NBV=NEQB
INVE  11          IF (NBV.EQ.NEQB) IND=1
INVE  12          NBVN=0
INVE  13          ICOUNT=0
INVE  14          LL=0
INVE  15    C
INVE  16          REWIND NMASS
INVE  17          REWIND NSTIF
INVE  18     60   READ (NMASS) XM
INVE  19          READ (NSTIF) VA
INVE  20          ICOUNT=ICOUNT+1
INVE  21          DO 20 I=1,NEQB
INVE  22          IF (VA(I).EQ.0.0) GO TO 20
INVE  23          VA(I)=XM(I)/VA(I)
INVE  24     20   CONTINUE
INVE  25    C
```

```
INVE  26          NNV=NEQB/NBV
INVE  27          DO 40 L=1,NBV
INVE  28          RT=0.0
INVE  29          NA=L*NNV
INVE  30          DO 34 I=1,NN
INVE  31          IF (VA(I).LT.RT) GO TO 34
INVE  32          RT=VA(I)
INVE  33          IJ=I
INVE  34     34   CONTINUE
INVE  35          DO 30 I=NN,NEQB
INVE  36          IF (VA(I).LE.RT) GO TO 30
INVE  37          RT=VA(I)
INVE  38          IJ=I
INVE  39     30   CONTINUE
INVE  40          IF (VA(IJ).NE.0.0) GO TO 32
INVE  41          NBVN=NBVN+1
INVE  42          GO TO 40
INVE  43     32   LL=LL+1
INVE  44          IEQ(LL)=(ICOUNT-1)*NEQB+IJ
INVE  45          IF (LL.GE.NV1) GO TO 50
INVE  46          VA(IJ)=0.0
INVE  47     40   CONTINUE
INVE  48          IF (IND.EQ.1) GO TO 45
INVE  49          IF ((NBVN.EQ.0).OR.(ICOUNT.EQ.NBLOCK)) GO TO 45
INVE  50          NBV=KK*((NV1-LL-1)/(NBLOCK-ICOUNT)+1)
INVE  51          IF (NBV.GT.NEQB) NBV=NEQB
INVE  52          NBVN=0
INVE  53    C
INVE  54     45   IF (ICOUNT.LT.NBLOCK) GO TO 60
INVE  55          IF (IND.EQ.1) GO TO 47
INVE  56          KK=2*KK
INVE  57          GO TO 90
INVE  58     47   PRINT 1000
INVE  59          STOP
INVE  60    C
INVE  61     50   REWIND NMASS
INVE  62          REWIND NR
INVE  63          DO 100 L=1,NBLOCK
INVE  64          READ (NMASS) XM
INVE  65          DO 120 I=1,NEQB
INVE  66          VA(I,1)=XM(I)
INVE  67          DO 120 J=2,NV
INVE  68    120   VA(I,J)=0.0
INVE  69          DO 140 K=2,NV
INVE  70          II=IEQ(K-1)
INVE  71          NLE=(L-1)*NEQB
INVE  72          NRI=L*NEQB
INVE  73          IF (II-NLE) 140,140,160
INVE  74    160   IF (NRI-II) 140,180,180
INVE  75    180   II=II-NLE
INVE  76          VA(II,K)=1.
INVE  77    140   CONTINUE
INVE  78          WRITE (NR) VA
INVE  79    100   CONTINUE
INVE  80          PRINT 1010
INVE  81          PRINT 1020,(IEQ(I),I=1,NV1)
INVE  82    C
INVE  83          RETURN
INVE  84    1000  FORMAT (42HOWE DO NOT HAVE ENOUGH FINITE EIGENVALUES  )
INVE  85    1010  FORMAT (20HOPRINT OF VECTOR IEQ  )
INVE  86    1020  FORMAT (1HC,20I6)
INVE  87          END
```

```
REDB   1          SUBROUTINE REDBAK (A,VA,VV,MAXB,NEQB,NV,NWA,NWV,NWVV,NTB,NBLOCK)
REDB   2    C
REDB   3          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
REDB   4          DIMENSION A(NWA),VA(NWV),VV(NWVV),MAXB(NEQB)
REDB   5    C
REDB   6          NEB=NTB*NEQB
```

```
REDB   7          NEPT=NEB+NEQB
REDB   8  C
REDB   9  C   REDUCE VECTORS ON TAPE NR
REDB  10          REWIND NRED
REDB  11          REWIND NR
REDB  12          REWIND NL
REDB  13          REWIND NT
REDB  14          READ (NRED) A,MAXB
REDB  15          ISV=NTB+1
REDB  16          LL=0
REDB  17          DO 10 L=1,ISV
REDB  18          READ (NR) VA
REDB  19          K=0
REDB  20          KK=LL
REDB  21          DO 20 J=1,NV
REDB  22          DO 30 I=1,NEQB
REDB  23          K=K+1
REDB  24          KK=KK+1
REDB  25    30    VV(KK)=VA(K)
REDB  26    20    KK=KK+NEB
REDB  27    10    LL=LL+NEQB
REDB  28          ISA=1
REDB  29  C
REDB  30   500    DO 100 I=1,NEQB
REDB  31          IL=I+NEQB
REDB  32          MAX=MAXB(I)
REDB  33          J=0
REDB  34          DO 120 II=IL,MAX,NEQB
REDB  35          J=J+1
REDB  36          C=A(II)
REDB  37          IF (C) 110,120,110
REDB  38   110    KK=I+J
REDB  39          JJ=I
REDB  40          DO 140 L=1,NV
REDB  41          VV(KK)=VV(KK)-C*VV(JJ)
REDB  42          KK=KK+NEBT
REDB  43   140    JJ=JJ+NEBT
REDB  44   120    CONTINUE
REDB  45   100    CONTINUE
REDB  46          DO 200 I=1,NEQB
REDB  47          C=A(I)
REDB  48          IF (C) 180,200,180
REDB  49   180    KK=I
REDB  50          DO 210 L=1,NV
REDB  51          VV(KK)=VV(KK)/C
REDB  52   210    KK=KK+NEBT
REDB  53   200    CONTINUE
REDB  54          IF (ISA.EQ.NBLOCK) GO TO 400
REDB  55          READ (NRED) A,MAXB
REDB  56          ISA=ISA+1
REDB  57  C
REDB  58  C   STORE REDUCED VECTORS ON TAPE NT
REDB  59          K=0
REDB  60          KK=0
REDB  61          DO 240 J=1,NV
REDB  62          DO 220 I=1,NEQB
REDB  63          K=K+1
REDB  64          KK=KK+1
REDB  65   220    VA(K)=VV(KK)
REDB  66   240    KK=KK+NEB
REDB  67          WRITE (NT) VA
REDB  68          K=1
REDB  69          DO 310 J=1,NV
REDB  70          DO 300 I=1,NEB
REDB  71          KK=K+NEQB
REDB  72          VV(K)=VV(KK)
REDB  73   300    K=K+1
REDB  74   310    K=K+NEQB
REDB  75          IF (ISV.EQ.NBLOCK) GO TO 500
REDB  76          READ (NR) VA
REDB  77          ISV=ISV+1
REDB  78          KK=NEB
REDB  79          K=0
REDB  80          DO 330 J=1,NV
REDB  81          DO 320 I=1,NEQB
REDB  82          K=K+1
REDB  83          KK=KK+1
REDB  84   320     VV(KK)=VA(K)
REDB  85   330     KK=KK+NEB
REDB  86          GO TO 500
REDB  87  C
REDB  88  C   BACKSUBSTITUTE VECTORS ON TAPE NT
REDB  89   400    BACKSPACE NRED
REDB  90          ISA=1
REDB  91   420    DO 600 I=1,NEQB
REDB  92          J=NEQB+1-I
REDB  93          MAX=MAXB(J)
REDB  94          IF (A(J)) 440,600,440
REDB  95   440    KK=J
REDB  96          DO 620 L=1,NV
REDB  97          JJ=KK+1
REDB  98          IL=J+NEQB
REDB  99          C=VV(KK)
REDB 100          DO 640 II=IL,MAX,NEQB
REDB 101          C=C-A(II)*VV(JJ)
REDB 102   640    JJ=JJ+1
REDB 103          VV(KK)=C
REDB 104   620    KK=KK+NEBT
REDB 105   600    CONTINUE
REDB 106          KK=0
REDB 107          K=0
REDB 108          DO 660 J=1,NV
REDB 109          DO 670 I=1,NEQB
REDB 110          K=K+1
REDB 111          KK=KK+1
REDB 112   670    VA(K)=VV(KK)
REDB 113   660    KK=KK+NEB
REDB 114          WRITE (NL) VA
REDB 115          IF (ISA.EQ.NBLOCK) GO TO 800
REDB 116          BACKSPACE NRED
REDB 117          READ (NRED) A,MAXB
REDB 118          BACKSPACE NRED
REDB 119          ISA=ISA+1
REDB 120          BACKSPACE NT
REDB 121          READ (NT) VA
REDB 122          BACKSPACE NT
REDB 123          K=NEBT
REDB 124          DO 700 J=1,NV
REDB 125          DO 720 I=1,NEB
REDB 126          KK=K-NEQB
REDB 127          VV(K)=VV(KK)
REDB 128   720    K=K-1
REDB 129   700    K=K+NEBT+NEB
REDB 130          K=0
REDB 131          KK=0
REDB 132          DO 740 J=1,NV
REDB 133          DO 760 I=1,NEQB
REDB 134          K=K+1
REDB 135          KK=KK+1
REDB 136   760    VV(KK)=VA(K)
REDB 137   740    KK=KK+NEB
REDB 138          GO TO 420
REDB 139   800    RETURN
REDB 140          END
```

```
EIGS   1          SUBROUTINE EIGSOL (DL,RTOLV,AR,BR,VEC,VL,VE,D,XM,NE,NV,NBLOCK,
EIGS   2         1NEQB,NITE)
EIGS   3  C
EIGS   4          COMMON /TAPES/NSTIF,NRED,NL,NR,NT,NMASS
EIGS   5          DIMENSION AR(NV,NV),BR(NV,NV),VEC(NV,NV),VL(NEQB,NV),VE(NEQB,NV)
EIGS   6          DIMENSION D(NV),DL(NV),RTOLV(NV),XM(NEQB)
EIGS   7  C
EIGS   8          NITEM=12
EIGS   9          RTOL=1.0E-06
```

```
EIGS  10          TOLJ=1.0E-12
EIGS  11          REWIND NMASS
EIGS  12          REWIND NT
EIGS  13          REWIND NR
EIGS  14  C
EIGS  15  C    FIND PROJECTIONS OF MASS AND STIFFNESS OPERATORS
EIGS  16          DO 100 I=1,NV
EIGS  17          DO 100 J=I,NV
EIGS  18          AR(I,J)=0.0
EIGS  19   100    BR(I,J)=0.0
EIGS  20          DO 200 N=1,NBLOCK
EIGS  21          BACKSPACE NL
EIGS  22          READ (NL) VL
EIGS  23          BACKSPACE NL
EIGS  24          READ (NR) VR
EIGS  25          READ (NMASS) XM
EIGS  26          DO 220 I=1,NV
EIGS  27          DO 220 J=I,NV
EIGS  28          ART=0.0
EIGS  29          DO 230 K=1,NEQB
EIGS  30   230    ART=ART+VL(K,I)*VR(K,J)
EIGS  31   220    AR(I,J)=AR(I,J)+ART
EIGS  32          DO 240 I=1,NEQB
EIGS  33          XMM=XM(I)
EIGS  34          DO 240 J=1,NV
EIGS  35   240    VR(I,J)=VL(I,J)*XMM
EIGS  36          WRITE (NT) VR
EIGS  37          DO 260 I=1,NV
EIGS  38          DO 260 J=I,NV
EIGS  39          BRT=0.0
EIGS  40          DO 280 K=1,NEQB
EIGS  41   280    BRT=BRT+VL(K,I)*VR(K,J)
EIGS  42   260    BR(I,J)=BR(I,J)+BRT
EIGS  43   200    CONTINUE
EIGS  44          DO 290 I=1,NV
EIGS  45          DO 290 J=1,I
EIGS  46          AR(I,J)=AR(J,I)
EIGS  47   290    BR(I,J)=BR(J,I)
EIGS  48  C
EIGS  49  C    SOLVE EIGENVALUE PROBLEM
EIGS  50          PRINT 1010
EIGS  51          DO 292 I=1,NV
EIGS  52   292    PRINT 1000,(AR(I,J),J=1,NV)
EIGS  53          PRINT 1020
EIGS  54          DO 294 I=1,NV
EIGS  55   294    PRINT 1000,(BR(I,J),J=1,NV)
EIGS  56  C
EIGS  57          CALL JACOBI (AR,BR,VEC,D,VL,NV,TOLJ)
EIGS  58          DO 295 J=1,NV
EIGS  59          XMM=SQRT(BR(J,J))
EIGS  60          DO 295 K=1,NV
EIGS  61   295    VEC(K,J)=VEC(K,J)/XMM
EIGS  62  C
EIGS  63          PRINT 1030
EIGS  64          PRINT 1010
EIGS  65          DO 296 I=1,NV
EIGS  66   296    PRINT 1000,(AR(I,J),J=1,NV)
EIGS  67          PRINT 1020
EIGS  68          DO 298 I=1,NV
EIGS  69   298    PRINT 1000,(BR(I,J),J=1,NV)
EIGS  70  C
EIGS  71  C    ARRANGE EIGENVALUES
EIGS  72          NV1=NV-1
EIGS  73   440    IS=0
EIGS  74          DO 400 I=1,NV1
EIGS  75          IF (D(I+1).GE.D(I)) GO TO 400
EIGS  76          IS=IS+1
EIGS  77          DT=BR(I+1,I+1)
EIGS  78          DT=D(I+1)
EIGS  79          BR(I+1,I+1)=BR(I,I)
EIGS  80          D(I+1)=D(I)
EIGS  81          BR(I,I)=BRT
EIGS  82          D(I)=DT
EIGS  83          DO 420 K=1,NV
EIGS  84          TEMP=VEC(K,I+1)
EIGS  85          VEC(K,I+1)=VEC(K,I)
EIGS  86   420    VEC(K,I)=TEMP
EIGS  87   400    CONTINUE
EIGS  88          IF (IS.GT.0) GO TO 440
EIGS  89  C
EIGS  90  C    CHECK FOR CONVERGENCE
EIGS  91          DO 300 I=1,NV
EIGS  92          DIF=ABS(DL(I)-D(I))
EIGS  93   300    RTOLV(I)=DIF/D(I)
EIGS  94          PRINT 1040
EIGS  95          PRINT 1000,(RTOLV(I),I=1,NV)
EIGS  96          DO 320 I=1,NF
EIGS  97          IF (RTOLV(I).GT.RTOL) GO TO 340
EIGS  98   320    CONTINUE
EIGS  99          PRINT 1050,RTOL
EIGS 100          NITE=NITEM
EIGS 101          GO TO 350
EIGS 102   340    IF (NITE.LT.NITEM) GO TO 360
EIGS 103          PRINT 1060
EIGS 104   350    DO 354 I=1,NV
EIGS 105   354    DL(I)=D(I)
EIGS 106          M=NT
EIGS 107          NT=NL
EIGS 108          NL=M
EIGS 109          GO TO 430
EIGS 110  C
EIGS 111  C    CALCULATE APPROXIMATE EIGEN DIRECTIONS
EIGS 112   360    DO 410 I=1,NV
EIGS 113   410    DL(I)=D(I)
EIGS 114   430    REWIND NT
EIGS 115          REWIND NR
EIGS 116          DO 460 N=1,NBLOCK
EIGS 117          READ (NT) VR
EIGS 118          DO 480 J=1,NV
EIGS 119          DO 480 I=1,NEQB
EIGS 120          TEMP=0.0
EIGS 121          DO 500 K=1,NV
EIGS 122   500    TEMP=TEMP+VR(I,K)*VEC(K,J)
EIGS 123   480    VL(I,J)=TEMP
EIGS 124   460    WRITE (NR) VL
EIGS 125  C
EIGS 126          RETURN
EIGS 127  1000    FORMAT (1H ,12E11.4)
EIGS 128  1002    FORMAT (1HC,6E20.14)
EIGS 129  1010    FORMAT (10HOMATRIX AR )
EIGS 130  1020    FORMAT (10HOMATRIX BR )
EIGS 131  1030    FORMAT (40HOAR AND BR AFTER JACOBI DIAGONALIZATION  )
EIGS 132  1040    FORMAT (32HOREL TOL REACHED ON EIGENVALUES  )
EIGS 133  1050    FORMAT (21HOCONVERGENCE FOR RTOL  E10.4)
EIGS 134  1060    FORMAT (31HOWE ACCEPT CURRENT ITERM VALUES  )
EIGS 135  1070    FORMAT (26HOTHE FINAL EIGENVALUES ARE  )
EIGS 136  1080    FORMAT (1H1,22HPRINT OF EIGENVECTORS  )
EIGS 137          END
```

```
JACO   1          SUBROUTINE JACOBI (A,B,X,EIGV,D,N,RTOL)
JACO   2  C
JACO   3          DIMENSION A(N,N),B(N,N),X(N,N),EIGV(N),D(N)
JACO   4  C
JACO   5          NSMAX=15
JACO   6          DO 10 I=1,N
JACO   7          D(I)=A(I,I)/B(I,I)
JACO   8   10     EIGV(I)=D(I)
JACO   9          IF (N.EQ.1) RETURN
JACO  10          DO 30 I=1,N
JACO  11          DO 20 J=1,N
JACO  12   20     X(I,J)=0.
JACO  13   30     X(I,I)=1.0
JACO  14          NSWEEP=0
JACO  15          NR=N-1
```

```
JACO  16  C
JACO  17  C     WE START ITERATION
JACO  18   40     NSWEEP=NSWEEP+1
JACO  19          PRINT 1000,NSWEEP
JACO  20          EPS=(0.01**NSWEEP)**2
JACO  21          DO 50 J=1,NR
JACO  22          JJ=J+1
JACO  23          DO 50 K=JJ,N
JACO  24          TT=A(J,K)*A(J,K)
JACO  25          TB=A(J,J)*A(K,K)
JACO  26          EPTOLA=ABS(TT/TB)
JACO  27          TT=B(J,K)*B(J,K)
JACO  28          TB=B(J,J)*B(K,K)
JACO  29          EPTOLB=TT/TB
JACO  30          IF ((EPTOLA.LT.EPS).AND.(EPTOLB.LT.EPS)) GO TO 50
JACO  31          AKK=A(K,K)*B(J,K)-B(K,K)*A(J,K)
JACO  32          AJJ=A(J,J)*B(J,K)-B(J,J)*A(J,K)
JACO  33          AB=A(J,J)*B(K,K)-A(K,K)*B(J,J)
JACO  34          CHECK=(AB*AB+4.0*AKK*AJJ)/4.0
JACO  35          IF (CHECK) 60,70,70
JACO  36   60     PRINT 1004,CHECK
JACO  37          STOP
JACO  38   70     SQCH=SQRT(CHECK)
JACO  39          D1=AB/2.0+SQCH
JACO  40          D2=AB/2.0-SQCH
JACO  41          DEN=D1
JACO  42          IF (ABS(D2).GT.ABS(D1)) DEN=D2
JACO  43          IF (DEN) 90,80,90
JACO  44   80     CA=0.
JACO  45          CG=-A(J,K)/A(K,K)
JACO  46          GO TO 100
JACO  47   90     CA=AKK/DEN
JACO  48          CG=-AJJ/DEN
JACO  49  C
JACO  50  C     WE PERFORM THE GENERALIZED ROTATION
JACO  51   100    IF (N-2) 95,180,95
JACO  52   95     JP1=J+1
JACO  53          JM1=J-1
JACO  54          KP1=K+1
JACO  55          KM1=K-1
JACO  56  C
JACO  57          IF (JM1-1) 120,110,110
JACO  58   110    DO 105 I=1,JM1
JACO  59          AJ=A(I,J)
JACO  60          BJ=B(I,J)
JACO  61          AK=A(I,K)
JACO  62          BK=B(I,K)
JACO  63          A(I,J)=AJ+CG*AK
JACO  64          B(I,J)=BJ+CG*BK
JACO  65          A(I,K)=AK+CA*AJ
JACO  66   105    B(I,K)=BK+CA*BJ
JACO  67  C
JACO  68   120    IF (KP1-N) 130,130,140
JACO  69   130    DO 125 I=KP1,N
JACO  70          AJ=A(J,I)
JACO  71          BJ=B(J,I)
JACO  72          AK=A(K,I)
JACO  73          BK=B(K,I)
JACO  74          A(J,I)=AJ+CG*AK
JACO  75          B(J,I)=BJ+CG*BK
JACO  76          A(K,I)=AK+CA*AJ
JACO  77   125    B(K,I)=BK+CA*BJ
JACO  78  C
JACO  79   140    IF (JP1-KM1) 150,150,180
JACO  80   150    DO 160 I=JP1,KM1
JACO  81          AJ=A(J,I)
JACO  82          BJ=B(J,I)
JACO  83          AK=A(I,K)
JACO  84          BK=B(I,K)
JACO  85          A(J,I)=AJ+CG*AK
JACO  86          B(J,I)=BJ+CG*BK
JACO  87          A(I,K)=AK+CA*AJ
JACO  88   160    B(I,K)=BK+CA*BJ
JACO  89   180    AK=A(K,K)
JACO  90          BK=B(K,K)
JACO  91          A(K,K)=AK+2*CA*A(J,K)+CA*CA*B(J,J)
JACO  92          B(K,K)=BK+2*CA*B(J,K)+...+CA*B(J,J)
JACO  93          A(J,J)=A(J,J)+2*CG*A(J,K)+CG*CG*...
JACO  94          B(J,J)=B(J,J)+2*CG*B(J,K)+CG*CG*...
JACO  95          A(J,K)=0.0
JACO  96          B(J,K)=0.0
JACO  97  C
JACO  98  C     UPDATE EIGENVECTORS
JACO  99          DO 190 I=1,N
JACO 100          XJ=X(I,J)
JACO 101          XK=X(I,K)
JACO 102          X(I,J)=XJ+CG*XK
JACO 103   190    X(I,K)=XK+CA*XJ
JACO 104  C
JACO 105   50     CONTINUE
JACO 106  C
JACO 107          DO 220 I=1,N
JACO 108   220    EIGV(I)=A(I,I)/B(I,I)
JACO 109          PRINT 1005
JACO 110          PRINT 1002,(EIGV(I),I=1,N)
JACO 111  C
JACO 112  C     CHECK FOR CONVERGENCE
JACO 113          DO 240 I=1,N
JACO 114          TOL=RTOL*D(I)
JACO 115          DIF=ABS(EIGV(I)-D(I))
JACO 116   240    IF (DIF.GT.TOL) GO TO 300
JACO 117  C
JACO 118  C       CHECK IF ALL OFF-DIAG ELEMENTS ARE SATISFACTORILY SMALL
JACO 119          EPS=RTOL**2
JACO 120          DO 260 J=1,NR
JACO 121          JJ=J+1
JACO 122          DO 260 K=JJ,N
JACO 123          TT=A(J,K)*A(J,K)
JACO 124          TB=A(J,J)*A(K,K)
JACO 125          EPSA=ABS(TT/TB)
JACO 126          TT=B(J,K)*B(J,K)
JACO 127          TB=B(J,J)*B(K,K)
JACO 128          EPSB=TT/TB
JACO 129          IF ((EPSA.LT.EPS).AND.(EPSB.LT.EPS)) GO TO 260
JACO 130          GO TO 300
JACO 131   260    CONTINUE
JACO 132  C
JACO 133          DO 310 I=1,N
JACO 134          DO 310 J=1,N
JACO 135          B(J,I)=B(I,J)
JACO 136   310    A(J,I)=A(I,J)
JACO 137          RETURN
JACO 138  C
JACO 139   300    DO 320 I=1,N
JACO 140   320    D(I)=EIGV(I)
JACO 141          IF (NSWEEP.LT.NSMAX) GO TO 40
JACO 142          DO 330 I=1,N
JACO 143          DO 330 J=1,N
JACO 144          B(J,I)=B(I,J)
JACO 145   330    A(J,I)=A(I,J)
JACO 146          RETURN
JACO 147  C
JACO 148   1000   FORMAT (1HC,14HNO OF SWEEP =  I4)
JACO 149   1002   FORMAT (1H ,12E11.4)
JACO 150   1004   FORMAT (8HCCHECK =  E20.14)
JACO 151   1005   FORMAT (24HOCURRENT EIGENVALUES ARE  )
JACO 152          END
```

```
SCHE   1          SUBROUTINE SCHECK (DL,RTOLV,A,XM,BJP,BLC,BUPC,NEIV,NWA,NEQN,
SCHE   2         1NBLOCK,NF,NV,SHIFT,NEI)
SCHE   3  C
SCHE   4          COMMON /TAPES/NSTIF,NREO,NL,NP,NT,NMASS
SCHE   5          DIMENSION A(NWA),XM(NEQN),BJP(NV),BLC(NV),BUPC(NV),DL(NV),
SCHE   6         1RTOLV(NV)
```

```
      INTEGER NEIV(NV)
C
      RTOL=1.0E-6
      ETOL=1.0E-2
C
      DO 100 I=1,NV
      BUP(I)=DL(I)*(1.0+ETOL)
  100 BLO(I)=DL(I)*(1.0-ETOL)
      NROOT=0
      DO 120 I=1,NE
  120 IF (RTOLV(I).LT.RTOL) NROOT=NROOT+1
      IF (NROOT.GE.1) GO TO 200
      PRINT 1010
      STOP
C
C     FIND UPPER BOUNDS ON EIGENVALUE CLUSTERS
  200 DO 240 I=1,NROOT
  240 NEIV(I)=1
      IF (NROOT.NE.1) GO TO 260
      BUPC(1)=BUP(1)
      LM=1
      L=1
      I=2
      GO TO 295
  260 L=1
      I=2
  270 IF (BUP(I-1).LE.BLO(I)) GO TO 280
      NEIV(L)=NEIV(L)+1
      I=I+1
      IF (I.LE.NROOT) GO TO 270
  280 BUPC(L)=BUP(I-1)
      IF (I.GT.NROOT) GO TO 290
      L=L+1
      I=I+1
      IF (I.LE.NROOT) GO TO 270
      BUPC(L)=BUP(I-1)
  290 LM=L
  295 IF (BUP(I-1).LE.BLO(I)) GO TO 300
      IF (RTOLV(I).GT.RTOL) GO TO 300
      BUPC(L)=BUP(I)
      NEIV(L)=NEIV(L)+1
      NROOT=NROOT+1
      IF (NROOT.EQ.NV) GO TO 300
      I=I+1
      GO TO 295
C
C     FIND SHIFT
  300 PRINT 1020
      PRINT 1005,(BUPC(I),I=1,LM)
      PRINT 1030
      PRINT 1006,(NEIV(I),I=1,LM)
      LL=LM-1
      IF (LM.EQ.1) GO TO 310
  330 DO 320 I=1,LL
  320 NEIV(L)=NEIV(L)+NEIV(I)
      L=L+1
      LL=LL-1
      IF (L.NE.1) GO TO 330
      PRINT 1040
      PRINT 1006,(NEIV(I),I=1,LM)
  310 DO 340 I=1,LM
      IF (NEIV(I).GE.NROOT) GO TO 350
  340 CONTINUE
  350 SHIFT=BUPC(I)
      NEI=NEIV(I)
C
C     SHIFT MATRIX
      REWIND NSTIF
      REWIND NMASS
      REWIND NRED
      DO 400 L=1,NBLOCK
      READ (NSTIF) A
      READ (NMASS) XM
      DO 420 I=1,NEQB
  420 A(I)=A(I)-SHIFT*XM(I)
      WRITE (NRED) A
  400 CONTINUE
      I=NSTIF
      NSTIF=NRED
      NRED=I
      RETURN
C
 1005 FORMAT (1HO,6E22.14)
 1006 FORMAT (1HO,6I22)
 1010 FORMAT (30HOCONVERGENCE FOR NO EIGENVALUE  )
 1020 FORMAT (37HOUPPER BOUNDS ON EIGENVALUE CLUSTERS  )
 1030 FORMAT (34HONO OF EIGENVALUES IN EACH CLUSTER  )
 1040 FORMAT (42HONO OF EIGENVALUES LESS THAN UPPER BOUNDS  )
      END
```