EIGEN SOLVERS
FOR STRUCTURAL PROBLEMS

Some algorithms for symmetric eigenvalue problems and their merits

K. Bell
EIGENSOLVERS FOR STRUCTURAL PROBLEMS

Some algorithms for symmetric eigenvalue problems and their merits
EIGENSOLVERS
FOR STRUCTURAL PROBLEMS

Some algorithms for symmetric eigenvalue problems and their merits

Kolbein Bell
Norwegian University of Science and Technology
Faculty of civil and environmental engineering
Department of structural engineering
currently on sabbatical leave at
Delft University of Technology
Faculty of civil engineering

Preface

This effort is long overdue since most of the work reported was carried out quite some time ago. However, the delay has enabled some fairly recent developments to be included, and above all it has enabled larger problems to be numerically investigated on much faster PC equipment.

I should point out right away that, although I have been interested in the symmetric matrix eigenvalue problem for a very long time, I am by no means an expert on the subject. I am an engineer, not a mathematician, and my interest has been in the solution of eigenvalue problems as they occur in the analysis of structural systems, notably in structural dynamics (free or natural vibration) and stability (linearized buckling). The computer implementation of the various algorithms, in the form of general, Fortran subroutines, has been my main concern.

During the past two decades this work has been associated with a local ('in-house') library of Fortran subroutines for finite element type analyses, the SAM library, which is organized in more or less independent and self-contained 'packages'. One such package contains subroutines for symmetric eigenvalue problems (eigensolvers), and all implementations described in this report belong to this package. In a limited sense this report may thus be viewed as a SAM library document, containing both theoretical background and numerical verification, as well as recommendations for the practical use of the library subroutines. This alone, however, would hardly justify a publication of this nature. In spite of its close relationship with a rather obscure library, the report is believed to contain information of interest to a wider readership. It does not offer much in terms of novelties, but some of the features of the computer implementations as well as the numerical performance and measured efficiency of the algorithms are thought to be of general interest.

Much of the work, including all programming, was carried out at the Norwegian University of Science and Technology (NTNU) in Trondheim, but the report itself has been written, and most of the actual computations have been carried out during my sabbatical leave at Delft University of Technology (TU Delft), Faculty of Civil Engineering. I would like to express my gratitude to both institutions; to my own university (NTNU) for granting me sabbatical leave, without which this report would probably never have been written, and to the Faculty of Civil Engineering at TU Delft for providing me with all necessary facilities as well as a friendly and stimulating environment. In particular I would like to thank Peter Stolle, of the timber structures group, for his always friendly and efficient efforts in keeping my PC/network/printer operating at all times.

Delft, March 1998

Kolbein Bell
# Content

## Part I Theory

### 1 Preliminaries

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Statement of problem</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Special matrices and matrix properties</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Some eigenvalue and eigenvector properties</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>Similarity transformations</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>The Rayleigh quotient and Rayleigh's principle</td>
<td>11</td>
</tr>
<tr>
<td>1.6</td>
<td>The Rayleigh-Ritz procedure</td>
<td>13</td>
</tr>
<tr>
<td>1.7</td>
<td>Gram-Schmidt orthogonalization</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Modified Gram-Schmidt orthogonalization</td>
<td>17</td>
</tr>
<tr>
<td>1.8</td>
<td>Sturm sequence check</td>
<td>18</td>
</tr>
<tr>
<td>1.9</td>
<td>The Krylov sequence and some of its properties</td>
<td>19</td>
</tr>
<tr>
<td>1.10</td>
<td>Change of problem - shift and inversion</td>
<td>21</td>
</tr>
<tr>
<td>1.11</td>
<td>Transformation to standard form</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Factorizing matrix A</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>Factorizing matrix B</td>
<td>24</td>
</tr>
</tbody>
</table>

### 2 Methods of Solution

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Polynomial iteration / determinant methods</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Determinant search</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Sign count methods</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>Transformation methods</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Jacobi diagonalization</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Givens tridiagonalization</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Householder tridiagonalization</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>The LR transformation</td>
<td>32</td>
</tr>
</tbody>
</table>
### Chapter 3: LARGE MATRIX EIGENSOLUTIONS

#### 3.1 Subspace iteration
- The basic algorithm
- Convergence
- Start vectors
- Computational procedure
- Solution of the reduced eigenproblem
- Speeding up the process

#### 3.2 Truncated Lanczos
- The Lanczos algorithm
- Reorthogonalization
- Starting vector
- Eigenvalue extraction and stop criteria
- Application to the generalized eigenproblem
- Some computational aspects

### Part II: Numerical results

#### 4 TEST PROCEDURE

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Purpose and scope of the numerical tests</td>
<td>61</td>
</tr>
<tr>
<td>4.2 Computational efficiency and accuracy</td>
<td>62</td>
</tr>
<tr>
<td>4.2.1 Efficiency</td>
<td>62</td>
</tr>
<tr>
<td>4.2.2 Accuracy</td>
<td>64</td>
</tr>
<tr>
<td>4.3 Description of test problem</td>
<td>64</td>
</tr>
<tr>
<td>4.4 Computer and programming aspects</td>
<td>68</td>
</tr>
<tr>
<td>4.4.1 Hardware</td>
<td>68</td>
</tr>
<tr>
<td>4.4.2 Software</td>
<td>68</td>
</tr>
</tbody>
</table>

#### 5 THE SPECIAL EIGENPROBLEM

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subroutine HQLS</td>
<td>72</td>
</tr>
<tr>
<td>Subroutine HQRIS</td>
<td>72</td>
</tr>
<tr>
<td>Subroutine SPSJAC</td>
<td>73</td>
</tr>
</tbody>
</table>
5.2 Accuracy and robustness 73
5.3 Computational efficiency 76
5.4 The matrix condition number 80
5.5 Concluding remarks 82

6 THE GENERALIZED EIGENPROBLEM
- Small and full matrices 83

6.1 The subroutines 84
Subroutine GENHQL 84
Subroutine GENJAC 84

6.2 Accuracy and robustness 85
6.3 Computational efficiency 88
6.4 Concluding remarks 90

7 THE GENERALIZED EIGENPROBLEM
- Large and sparse matrices 91

7.1 The subroutines 91
Subroutine MSSIT (SPRSIT) 93
Subroutine LANCZ2 (SPRLAN) 94

7.2 Subspace iteration - MSSIT 95
Accuracy and robustness 96
Computational effort 100
Iteration or trial vectors 104
Speed up 108

7.3 Truncated Lanczos - LANCZ2 110
Accuracy and robustness 111
Reorthogonalization 118
Computational effort 123

7.4 Comparison between Lanczos and subspace iteration 127
7.5 Storage arrangement - skyline versus sparse 131
7.6 Conclusions and recommendations 133

References 137

Index 139
PART II

NUMERICAL RESULTS

TEST PROCEDURE

4.1

4.2

4.3

4.4

THE SPECIAL EIGENVALUE PROBLEM

References
In displacement based finite element analyses of structures, the symmetric eigenvalue problem is encountered in basically two areas, structural dynamics and linearized buckling.

In matrix notation the problem of free, undamped vibration of a structural system may be expressed as

\[(K - \omega^2 M)x = 0\]  

where \(K\) and \(M\) are the stiffness and mass matrices, respectively, of the structural (finite element) model, and \(\omega\) and \(x\) are the frequency and shape (in the form of displacement amplitudes) of the vibration mode(s).

The linearized buckling problem may be expressed as

\[(K - \rho K_G)x = 0\]  

where \(K\) is the stiffness matrix and \(K_G\) the so-called geometrical stiffness matrix of the structural model subjected to a given external loading. \(\rho\) is a load factor (buckling factor) and \(x\) is the corresponding shape (again in the form of displacement amplitudes) into which the structure 'buckle' under the influence of the external loading times \(\rho\). Normally \(K\) is taken to be the ordinary (linear) stiffness matrix, but in a slightly more refined analysis the external loading may be split into a constant part (e.g. dead load) and a variable part, and the geometric stiffness corresponding to the constant load is subtracted from the ordinary linear stiffness to give \(K\) in (b). In this case \(K_G\) in (b) only accounts for the variable part of the loading.

All system matrices in (a) and (b) are symmetric matrices, and both equations may, in 'neutral' notation, be written as

\[(A - \lambda B)q = 0\]  

where \(A\) and \(B\) are symmetric matrices. This is the mathematical statement of a generalized, symmetric eigenvalue problem, where \(\lambda\) is the eigenvalue and \(q\) the corresponding eigenvector.

The system matrices, which may be quite large, have some important properties. Apart from symmetry, they are also sparsely popu-
lated (the portion of non-zero terms is often very small), and this may, through various schemes of reordering, be exploited in terms of both storage and number of operations carried out during numerical computations.

Without going into details here we mention that the ordinary (linear) stiffness matrix $K$ is always positive definite.

The mass matrix $M$ depends on the mass model, of which we have two or perhaps three candidates: (1) the so-called consistent mass model, in which case $M$ has the same form and shape as $K$ (they have non-zero elements in the same locations), and it is also positive definite; (2) the lumped mass model, in which case $M$ is a diagonal matrix with (usually) some zero entries on the diagonal - this matrix is positive semi-definite; (3) the diagonalized mass model, in which case $M$ is a diagonal matrix with all diagonal entries being non-zero, positive numbers, rendering $M$ positive definite.

The geometric stiffness matrix $K_G$ has the same shape and form as $K$. In general, however, it is an indefinite matrix. The implications of the 'definite' property of the matrices will become evident in the chapters dealing with the theoretical aspects of the symmetric eigenvalue problem.

For many real problems the complete solution of (a) or (b) would represent a formidable numerical task. In most cases, however, we only require a few or a relatively small number of eigenvalues and corresponding eigenvectors to be determined, usually at the low end of the spectrum. Hence, the primary concern of this report is to develop accurate, robust and efficient algorithms for the partial solution of (c), and to implement these algorithms in the form of subprograms to be included in an 'in-house' library of Fortran subroutines for finite element type structural analysis problems, the SAM library [17]. The subroutines should be able to handle the free vibration problem (with different mass matrix models) and the linearized buckling problem described above, and while efficiency is important, robustness and ease of use are also important design criteria.

The subroutines are developed on the basis that all matrices involved can be stored simultaneously in primary computer storage (which need not all be physical RAM providing the operating system can handle virtual memory). Depending on the computer platform (hardware configuration and operating system) this may pose some restrictions on the size of the problems that can be handled, but the trends in computer development seem to suggest that these limitations become less and less serious. It should also be emphasized that efficient storage schemes are adopted, and versions of the main subroutines are available in both so-called skyline storage format, and the even more efficient sparse storage format. Combined
with efficient reordering schemes this enable quite large problems to be solved, even on quite inexpensive PC equipment.

The report is divided into two main parts: PART I deals with the theoretical aspects in three chapters. Chapter 1 defines the problem and summarizes some basic concepts that are central in its solution. Change of problem (through shift and inversion) and transformation from general to special form are also discussed. The chapter does not attempt a rigorous mathematical treatment, and while some parts are very sketchy others are more comprehensive (but not necessarily mathematically ‘complete’).

Chapter 2 summarizes some of the most important computational methods used for solving the symmetric eigenvalue problem (or just eigenproblem). The emphasis is on methods for complete solution of ‘small’ problems involving basically full matrices (matrices for which all \( n \) by \( n \) elements are stored). The two major methods for the problem in (c) involving large, sparse matrices, the subspace iteration method and the truncated Lanczos method, are developed in some detail in Chapter 3, including comments on computer implementation. The proposed implementations are believed to be reliable, accurate and robust and fairly efficient, without claiming to be to the fastest possible.

PART I, which is based on lecture notes from an advanced course on structural dynamics, serves as a ‘theory manual’ for the eigensolvers of the SAM library.

PART II presents, in four chapters, a fairly comprehensive set of results from numerical computations, all of which are related to free, undamped vibration of 3D skeletal frames. All computations have been carried out on (four different) PCs, and computational efficiency is based on actual, ‘measured’ CPU-times.

Chapter 4 outlines the purpose and scope of the numerical tests carried out, and it describes in some detail the free vibration problem used in the tests, with the aim of enabling anyone to reproduce the results.

Chapter 5 is devoted to numerical testing of three subroutines for the solution of the special eigenproblem, involving a full matrix. Matrices with a dimension in excess of 1000 are considered. A simplified ‘formula’ for computing the condition number of a matrix is also tested in this chapter.

Chapter 6 examines the performance of two algorithms for the complete solution of small, generalized eigenproblems involving full matrices.

The main eigensolvers of the SAM library, based on subspace iteration and truncated Lanczos, respectively, are tested and compared in Chapter 7. In addition to demonstrate both accuracy and robust-
ness, the results also show how the total computational effort is divided between the various sub-operations, for each subroutine, and how the two algorithms compare with respect to computational efficiency and storage requirements. One section also examines the difference between the two storage formats, 'skyline' and 'sparse'.

PART II serves as validation of the SAM library eigensolvers, and it provides the users of these subroutines with some guidelines as to which options should be used under different circumstances. Hopefully these chapters also contain some useful information for users of symmetric eigensolvers in general.
PART I

Theory
PRELIMINARIES

This chapter defines the problem and summarizes some basic properties and concepts that are central in its solution. This includes similarity transformations, the Rayleigh quotient and the Rayleigh-Ritz procedure. Gram-Schmidt orthogonalization is reviewed and an important sign count property, the Sturm sequence check, is stated. The Krylov sequence is defined and some of its important properties demonstrated. Finally, change of problem, through shift and inversion, and transformation from general to special form are discussed.

1.1 Statement of problem

Consider two real\(^1\), symmetric \(n\) by \(n\) matrices \(A\) and \(B\). The equation

\[(A - \lambda B)q = 0\]  

or

\[Aq = \lambda Bq\]

defines the generalized symmetric eigenproblem. If \(B\) is the identity matrix, that is \(B = I\), the above equations reduce to

\[(A - \lambda I)q = 0\]

and

\[Aq = \lambda q\]

---

\(^1\) Unless otherwise stated, our matrices are always real.
respective, which define the special or standard symmetric eigen-problem. In these equations the scalar $\lambda$ represents the eigenvalue and $q$ the corresponding eigenvector.

The standard eigenproblem, (1.2), has $n$ non-trivial solutions. The $i$'th solution is given by the eigenpair $(\lambda_i, q_i)$ for which we have

$$A q_i = \lambda_i q_i$$  \hspace{1cm} (1.3)

We order the eigenvalues such that

$$\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_i \leq \ldots \leq \lambda_n$$

In order for Equation (1.2a) to have non-trivial solutions we must have

$$\det(A - \lambda I) = 0$$  \hspace{1cm} (1.4)

and the eigenvalues are the roots of the characteristic polynomial

$$p(\lambda) = \det(A - \lambda I) = 0$$  \hspace{1cm} (1.5)

which has $n$ roots, not necessarily distinct.

Let

$$\Lambda = \begin{bmatrix} \lambda_1 & \lambda_2 & \ldots & \lambda_i & \ldots & \lambda_n \end{bmatrix}$$  \hspace{1cm} (1.6)

be a diagonal matrix of all eigenvalues, and

$$Q = [q_1, q_2, \ldots, q_i, \ldots, q_n]$$  \hspace{1cm} (1.7)

be a square matrix whose columns are the corresponding eigenvectors, that is, $(\lambda_i, q_i)$ is an eigenpair.

Equation (1.2b) may now be written as

$$AQ = QA$$  \hspace{1cm} (1.8)

and (1.1b) as

$$AQ = BQ\Lambda$$  \hspace{1cm} (1.9)

### 1.2 Special matrices and matrix properties

Some matrices with special patterns of zero entries, with special symmetries and other special properties are summarized in this section for reference.

The *unit* or identity matrix is always denoted by $I$, that is

$$I = [\delta_{ij}] = [e_1, e_2, \ldots, e_i, \ldots, e_n]$$  \hspace{1cm} (1.10)

where $\delta_{ij}$ is the Kronecker delta with the properties
Definitions

Let $G_{ij}$ be a typical element of a square matrix $G$, that is, $G = [G_{ij}]$. We say that $G$ is

- **diagonal** if $G_{ij} = 0$ whenever $i \neq j$
- **tridiagonal** if $G_{ij} = 0$ whenever $|i - j| > 1$
- **upper triangular** if $G_{ij} = 0$ whenever $i > j$
- **lower triangular** if $G_{ij} = 0$ whenever $i < j$
- **upper Hessenberg** if $G_{ij} = 0$ whenever $i > j+1$.

A triangular matrix is **unit triangular** if it has ones on its diagonal.

The transpose of matrix $G$ is denoted by $G^T$. Let $G$ be a square $n$ by $n$ matrix and $x$ and $y$ arbitrary $n$ by $1$ vectors.

**Matrix $G$** is

- **symmetric** if $G^T = G$
- **skew-symmetric** if $G^T = -G$
- **positive definite** if $x^T G x > 0$ for all $x \neq 0$
- **positive semi-definite** if $x^T G x \geq 0$ for all $x \neq 0$
- **indefinite** if $(x^T G x)(y^T G y) < 0$ for some $x$ and $y$
- **orthogonal** if $G^T G = I$
- **diagonal dominant** if $|G_{ii}| > \sum_{j \neq i} |G_{ij}|$ for all $i$
- **singular** if det $G = 0$

A symmetric, non-singular matrix $A$ may always be factorized or decomposed such that

$$A = L A D A L_A^T$$
(1.11)

where $L_A$ is a unit lower triangular matrix and $D_A$ is a diagonal matrix. If $A$ is also positive definite, all entries of $D_A$ are greater than zero. Hence $D_A$ can be expressed as

$$D_A = D_A^{1/2} D_A^{1/2}$$

and the decomposition of $A$ may be written as

$$A = U_A^T U_A$$
(1.12)

---

1. It should be noted that unless otherwise stated a vector is always assumed to be a column vector (or matrix). A row vector will be designated by a superscript $T$ on the vector symbol (it is the transpose of a column vector).
where $\mathbf{U}_A = \mathbf{D}_A^{1/2} \mathbf{L}_A^T$ is an upper triangular matrix. The decomposition (1.12) is attributed to Cholesky, and we shall refer to $\mathbf{U}_A$ as the Cholesky factor of $\mathbf{A}$. In order to distinguish between the two types of decomposition, we shall henceforth always use $\mathbf{U}$ for the (upper) triangular matrix in the Cholesky decomposition, whereas the triangular factor of the more general decomposition (1.11) will be designated by $\mathbf{L}$, which is a lower unit triangular matrix.

It should be noted that a special positive semi-definite matrix, namely a diagonal matrix with some zero elements on the diagonal (such as a lumped mass matrix), can also be decomposed according to (1.12). However, its factor will not be a true Cholesky factor, since it does not have an inverse.

### 1.3 Some eigenvalue and eigenvector properties

In this section we summarize, without proof, some important properties of eigenvalues and eigenvectors of real, symmetric matrices. For proofs, the reader is referred to one of the many excellent textbooks on the subject, e.g., PARLETT [1], GOLUB and VAN LOAN [2] or JENNINGS [3].

1. All eigenvalues of a symmetric matrix are real.
2. All eigenvalues of a positive definite matrix are positive ($> 0$).
3. The sum of the eigenvalues of a matrix is equal to the trace of the matrix, that is
   \[ \text{tr}(\mathbf{A}) = A_{11} + A_{22} + \ldots + A_{nn} = \lambda_1 + \lambda_2 + \ldots + \lambda_n \]  
   \[ (1.13) \]
4. The product of the eigenvalues of a matrix equals the determinant of the matrix, that is
   \[ \det(\mathbf{A}) = \lambda_1 \lambda_2 \lambda_3 \ldots \lambda_n \]  
   \[ (1.14) \]

From this it follows that a singular matrix must have at least one zero eigenvalue.
5. Since the determinant of a triangular matrix is simply the product of the diagonal elements, it follows that the eigenvalues of a triangular, and hence also a diagonal, matrix are equal to the diagonal elements.
6. Consider two eigenpairs of a symmetric matrix $\mathbf{A}$, that is $(\lambda, \mathbf{q})$ and $(\lambda', \mathbf{q}')$, where $\lambda$ and $\lambda'$ are distinct ($\lambda \neq \lambda'$). Then
   \[ \mathbf{A} \mathbf{q}_i = \lambda_i \mathbf{q}_i \quad \text{and} \quad \mathbf{A} \mathbf{q}_j = \lambda_j \mathbf{q}_j \]
Eigenvector orthogonality

Manipulating these two equations (premultiplication by $q_j^T$ and $q_i^T$, respectively, transposing and then subtracting) we find that

$$(\lambda_i - \lambda_j)q_j^T q_j = 0$$

Hence, since $\lambda_i \neq \lambda_j$,

$$q_i^T q_i = 0$$

If the eigenvectors are normalized such that

$$q_i^T q_i = 1$$

the orthogonality condition can be expressed as

$$q_i^T q_j = \delta_{ij} \quad \text{or} \quad Q^T Q = I \quad (1.15)$$

In deriving (1.15) it was assumed that the eigenvalues were distinct. It can be shown that for a symmetric matrix it is always possible to establish a set of $n$ orthonormal eigenvectors. However, the eigenvectors corresponding to multiple eigenvalues are not unique. With this reservation (1.15) is valid for all symmetric matrices, and we shall henceforth assume that the eigenvectors of the special eigenproblem are scaled such that (1.15) is satisfied.

For the generalized eigenproblem the eigenvectors are usually orthonormalized with respect to matrix $B$, that is

$$Q^T B Q = I \quad (1.16)$$

Eigenvectors satisfying (1.16) are said to be $B$-orthonormal.

From (1.8) and (1.15) it follows that

$$Q^T A Q = \Lambda \quad (1.17)$$

and

$$A = Q \Lambda Q^T \quad (1.18)$$

Equation (1.18) is referred to as the spectral decomposition of $A$.

It should be noted that while (1.15) is a necessary condition for the eigenvectors of a symmetric matrix, it is not a sufficient condition.
1.4 Similarity transformations

We consider the special eigenproblem in the form of Eq. (1.8), where \( A \) is a symmetric \( n \) by \( n \) matrix. Let \( N \) be a non-singular \( n \) by \( n \) matrix, and \( N^{-1} \) its inverse, that is

\[
NN^{-1} = I
\]  

Premultiplying Eq. (1.8) by \( N^{-1} \) gives

\[
N^{-1}AQ = N^{-1}QA
\]

or, in view of (1.19),

\[
N^{-1}ANN^{-1}Q = N^{-1}QA
\]

which may be written

\[
\tilde{A}Q = \tilde{Q}\Lambda
\]

where

\[
\tilde{A} = N^{-1}AN
\]

and

\[
\tilde{Q} = N^{-1}Q
\]

Hence the eigenvalues of \( \tilde{A} \) and \( A \) are the same. We say that these matrices are similar, and the transformation defined by (1.21) is called a similarity transformation. The eigenvectors of \( \tilde{A} \) are related to those of \( A \) through (1.22).

In order to preserve symmetry only orthogonal transformation matrices, for which

\[
N^{-1} = N^T
\]

are considered. Hence

\[
\tilde{A} = N^TAN
\]

which is always symmetric provided \( A \) is symmetric, and

\[
\tilde{Q} = N^TQ
\]

An interesting alternative way of performing a similarity transformation can be accomplished if the original matrix \( A \) can be factorized into a left, \( F_L \), and a right, \( F_R \), factor:

\[
A = F_LF_R
\]
THEORY

or

\[ F_L^{-1}A = F_R \]  \hfill (1.26)

Multiplying the factors in reversed order is equivalent to a similarity transformation:

\[ F_R F_L = F_L^{-1} A F_L \]  \hfill (1.27)

Similarity transformations form the basis of many important solution techniques.

1.5 The Rayleigh quotient and Rayleigh's principle

Consider the generalized eigenproblem of (1.1b) or (1.9). The Rayleigh quotient is defined as

\[ \rho(x) = \frac{x^T A x}{x^T B x} \]  \hfill (1.28)

where \( x \neq 0 \) is any vector in \( n \)-dimensional space, that is

\[ x = \sum_{i=1}^{n} c_i q_i = Qc \]  \hfill (1.29)

For the special eigenproblem (\( B = I \)) the scalar \( \rho(x) \) simplifies to

\[ \rho(x) = \frac{x^T A x}{x^T x} \]  \hfill (1.30)

From (1.28) and (1.29)

\[ \rho(x) = \frac{c^T Q^T A Q c}{c^T Q^T B Q c} = \frac{c^T A c}{c^T c} = \frac{\sum c_i^2 \lambda_i}{\sum c_i^2} \]  \hfill (1.31)

It is assumed that the eigenvectors are \( B \)-orthonormal, i.e., they satisfy Eq. (1.16).

If \( x \) is normalized so that

\[ x^T B x = c^T Q^T B Q c = c^T c = 1 \]  \hfill (1.32)

then

\[ \rho(x) = \sum_{i=1}^{n} c_i^2 \lambda_i = \lambda_a + \sum_{i=1}^{n} c_i^2 (\lambda_i - \lambda_a) \]  \hfill (1.33)
PART I

Assuming

\[ \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_r \leq \ldots \leq \lambda_n \]

it follows that

\[ \rho(x) - \lambda_i \geq 0 \quad \text{and} \quad \rho(x) - \lambda_n \leq 0 \]

Hence

\[ \lambda_1 \leq \rho(x) \leq \lambda_n \]  \hspace{1cm} (1.34)

The minimum value of \( \rho(x) \),

\[ \min \rho(x) = \lambda_1 \]

is reached when \( c = e_1 \) or \( x = Qe_1 = q_1 \), which also follows directly from (1.28). In general

\[ \rho(q_i) = \lambda_i \]  \hspace{1cm} (1.35)

If \( x \) is made orthogonal to \( q_1 \), that is

\[ q_1^T B x = e_1^T Q^T B Q e_1 = e_1^T c = c_1 = 0 \]

the inequality (1.34) becomes

\[ \lambda_2 \leq \rho(x) \leq \lambda_n \]

and the new minimum is attained when \( c = e_2 \) or \( x = q_2 \). Extending this idea we may state a property known as the recursive characterization of eigenvalues. A similar, but more general principle is the minimax characterization of eigenvalues, due to Courant. This principle states that

\[ \lambda_r = \max \left\{ \min_{x} \frac{x^T A x}{x^T B x} \right\} \quad r = 1, \ldots, n \]  \hspace{1cm} (1.36)

with \( x \) satisfying the restrictions \( x^T y_i = 0 \), for \( i = 1, \ldots, r-1 \).

In (1.36) we choose vectors \( y_i \) \( (i = 1, \ldots, r-1) \) and evaluate the minimum of \( \rho(x) \) with \( x \) subject to the restriction that it is orthogonal to \( y_i \). After having calculated this minimum we vary vectors \( y_i \) and find a new minimum, and so on. The maximum value that the minima reach is \( \lambda_r \).

This principle is the mathematical interpretation of Rayleigh's theorem: If a constraint is added to a vibrating system, its natural frequencies will never be lower than those of the original unconstrained system.

Assume that \( x \) is an approximation to the eigenvector \( q_i \), say

\[ x = q_i + \varepsilon y \]
THEORY

then the Rayleigh quotient of \( x \) will give an approximation to \( \lambda \), of order \( \varepsilon^2 \), that is

\[
p(x) = \lambda + O(\varepsilon^2)
\]  

(1.37)

Rayleigh's principle

This is a statement of Rayleigh's principle: A first order error in the eigenvector yields only a second order error in the corresponding eigenvalue.

1.6 The Rayleigh-Ritz procedure

Consider the generalized symmetric eigenproblem in the form of (1.9), that is

\[
A Q = B Q \lambda
\]

where \( A \) and \( B \) are symmetric \( n \times n \) matrices.

We establish a set of \( p \) linearly independent \( n \)-dimensional vectors \( w_1, w_2, \ldots, w_p \) which form the columns of the rank-\( p \) matrix \( W \).

These vectors will be referred to as Ritz basis vectors.

Question: Which linear combination of the Ritz basis vectors, expressed as

\[
V_i = W g_i \quad \text{or} \quad V = W g
\]

(1.38)

is the best approximation to the \( p \) first eigenvectors \( q_1, \ldots, q_p \) of the basic eigenproblem?

To answer this question we invoke the Rayleigh minimum principle by seeking a minimum of the Rayleigh quotient

\[
\rho(V_i) = \frac{v_i^T A v_i}{v_i^T B v_i} = \frac{g_i^T A g_i}{g_i^T B g_i}
\]

(1.39)

where

\[
\bar{A} = W^T A W \quad \text{and} \quad \bar{B} = W^T B W
\]

(1.40)

are symmetric \( p \times p \) matrices.

The Rayleigh quotient of (1.39) has a minimum if \( \mathbf{g}_i \) is an eigenvector of the reduced problem

\[
\bar{A} g_i = \omega_i \bar{B} g_i
\]

(1.41)

Ritz coordinates

The unknown Ritz coordinates, \( \mathbf{g}_i \), are thus obtained by solving the reduced \((p \times p)\) eigenproblem of (1.41). The corresponding eigenvalues, \( \omega_i \), are called Ritz values.
The complete solution of

\[ \bar{AG} = \bar{BG}\Omega \]  

(1.42)

which is an alternative way of writing (1.41), may be considered equivalent to minimizing the generalized (matrix) Rayleigh quotient

\[ \rho(V) = \frac{G^T W^T A W G}{G^T W^T B W G} = \frac{G^T \bar{A} G}{G^T \bar{B} G} \]  

(1.43)

The stationary value can be determined by

\[ \frac{\partial}{\partial G} \rho(V) = \frac{2 \bar{A} G (G^T \bar{B} G) - 2 \bar{B} G (G^T \bar{A} G)}{(G^T \bar{B} G)(G^T \bar{B} G)} = 0 \]

which, in view of

\[ (G^T \bar{A} G)(G^T \bar{B} G)^{-1} = \Omega \]

can be written as

\[ \frac{2(\bar{A} G - \bar{B} G \Omega)}{G^T \bar{B} G} = 0 \]  

(1.44)

The solution of (1.44) is equivalent to the solution of (1.42).

The Ritz vectors

\[ V = W G \]

thus represent the “optimal” estimate of the \( p \) first eigenvectors \( q_1, \ldots, q_p \) that can be derived from the basis vectors \( W \).

Providing \( G \) has been made \( \bar{B} \)-orthonormal we have

\[ V^T B V = G^T W^T B W G = G^T \bar{B} G = I \]  

(1.45)

In other words, \( V \) satisfies \( \bar{B} \)-orthonormality even if \( W \) does not.

The Ritz values

\[ \Omega = [\omega_1 \omega_2 \ldots \omega_p] \]

are estimates of the \( p \) first eigenvalues \( \lambda_i \). Since the Rayleigh-Ritz procedure can be regarded as adding constraints to the system - we have implicitly imposed the constraints \( w_{p+1} = \ldots = w_n = 0 \) - then by the max-min principle, which implies that eigenvalues will be raised by the imposition of constraints, the Ritz values will provide upper bounds such that

\[ \lambda_i \leq \omega_i \quad \text{for} \quad i = 1, 2, \ldots, p \]
1.7 Gram-Schmidt orthogonalization

Consider a system of \( m \) linearly independent, \( n \)-dimensional vectors \( \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \).

It is now possible to construct a set of \( n \)-dimensional, mutually orthogonal unit vectors.

\( \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m \)

satisfying

\[
\mathbf{u}_i^\top \mathbf{u}_j = \delta_{ij} \quad (i \leq m \text{ and } j \leq m)
\]

such that for each \( j = 1, 2, \ldots, m \)

\( \mathbf{u}_j \) is a linear combination of \( \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_j \) (1.47)

(proving that \( \mathbf{x}_j \) is also linearly independent.)

PROOF:

Since \( \mathbf{x}_1 \neq \mathbf{0} \) we have \( \mathbf{x}_1^\top \mathbf{x}_1 > 0 \) and \( \| \mathbf{x}_1 \| = \sqrt{\mathbf{x}_1^\top \mathbf{x}_1} > 0 \). The notation \( \| \mathbf{x} \| \) designates the length of the vector \( \mathbf{x}_1 \), also called the Euclidian norm or the 2-norm of \( \mathbf{x}_1 \).

We start with

\( \mathbf{u}_1 = \mathbf{x}_1 / \| \mathbf{x}_1 \| \)

For any \( k \geq 1 \) suppose we have constructed orthonormal vectors \( \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k \) \((k < m)\) such that (1.46) holds for all \( j \leq k \).

Next we construct \( \mathbf{u}_{k+1} \) such that (1.47) holds for all \( j \leq k + 1 \).

Let \( \mathbf{v} \) be a vector of the form

\[
\mathbf{v} = \mathbf{x}_{k+1} - \alpha_1 \mathbf{u}_1 - \alpha_2 \mathbf{u}_2 - \ldots - \alpha_k \mathbf{u}_k
\]

(1.48)

Since \( \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k \) are linear combinations of \( \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \), we have

\[
\mathbf{v} = \mathbf{x}_{k+1} + \text{lin. comb. of } \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k
\]

Hence \( \mathbf{v} \neq \mathbf{0} \), since \( \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \) are linearly independent. We now choose the coefficients \( \alpha_1, \alpha_2, \ldots, \alpha_k \) such that \( \mathbf{v} \) is orthogonal to \( \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k \). For \( j \leq k \) we have, in view of (1.46):

\[
\mathbf{u}_j^\top \mathbf{v} = \mathbf{u}_j^\top \mathbf{x}_{k+1} - \alpha_j = 0
\]

Thus, the non-zero vector \( \mathbf{v} \) is orthogonal to all unit vectors \( \mathbf{u}_j \) \((j \leq k)\) if we define

\[
\alpha_j = \mathbf{u}_j^\top \mathbf{x}_{k+1}
\]

(1.49)
To make $v$ a unit vector we divide it by its length

$$u_{k+1} = v/\|v\|$$  (1.50)

Relations (1.46) and (1.47) now hold for all $j \leq k + 1$, and the proposition follows by induction.

The process defined by (1.48) and (1.49) purges $x_{k+1}$ of all nonorthogonal components.

The Gram-Schmidt procedure may also be used to construct a system of vectors which are orthonormal with respect to a matrix.

Again we start with $m$ linearly independent vectors $z_1, \ldots, z_m$. Then we set

$$p_1 = \frac{z_1}{(z_1^TBz_1)^{1/2}}$$  (1.51)

Clearly $p_1^TBp_1 = 1$. Again we suppose that, for any $k \geq 1$, we have constructed $B$-orthonormal vectors $p_1, \ldots, p_k$, and we proceed to construct a $B$-orthonormal vector $p_{k+1}$ such that $p_1, \ldots, p_{k+1}$ are $B$-orthonormal vectors which are linear combinations of $z_1, \ldots, z_{k+1}$. Let

$$v = z_{k+1} - \sum_{i=1}^{k} \alpha_i p_i$$  (1.52)

Premultiplying this equation by $p_i^TB$ gives

$$p_i^TBv = p_i^TBz_{k+1} - \alpha_i \quad j = 1, \ldots, k$$

For $v$ to be $B$-orthogonal to all vectors $p_j (j \leq k)$, we require the left-hand side to be zero, which gives

$$\alpha_j = p_i^TBz_{k+1}$$  (1.53)

Hence

$$p_{k+1} = \frac{v}{(v^TBv)^{1/2}}$$  (1.54)

The "standard" Gram-Schmidt process - Eqs. (1.48) to (1.50) - may be written as follows:

For $i = 1, 2, \ldots, m$:

$$\hat{u}_i = x_i - \sum_{j=1}^{i-1} (u_j^T x_i) u_j \quad (\hat{u}_1 = x_1)$$  (1.55)
THEORY

Introducing the matrices
\[ X = [x_1, x_2, \ldots, x_m], \quad U = [u_1, u_2, \ldots, u_m] \]
and the upper triangular matrix
\[ R = [r_{ij}] \]
where \( r_{ij} = 0, \quad r_{ii} = \|u_i\| \) for all \( i \leq j \), Eq. (1.55) can be expressed in compact (matrix) form as
\[ X = UR \]
(1.56)
The right-hand side is an orthogonal decomposition of \( X \).

Modified Gram-Schmidt orthogonalization

The standard Gram-Schmidt algorithm (1.55) is prone to cancella-
tions and may produce vectors that are not quite orthogonal. A
numerically more stable yet mathematically equivalent procedure is
the modern version of the algorithm known as the modified Gram-
Schmidt process. It can be implemented by a slightly different
sequence from that given in (1.55):

For \( i = 1, 2, \ldots, m \):
\[ u_i = \frac{\hat{u}_i}{\|\hat{u}_i\|} \]
\[ x_{j}^{(i+1)} = x_{j}^{(i)} - (u_j^T x_{j}^{(i)}) u_i, \quad j = i+1, \ldots, m \]
(1.57)

where \( x_j^{(1)} = x_j \). The number of operations is exactly the same in
both cases. The major difference between the two algorithms is that
the modified Gram-Schmidt requires the vectors \( x_j (j = i+1, \ldots, m) \)
as well as the vectors \( u_k (k = i, i+1, \ldots) \) to be made orthogonal to
\( u_i \) as soon as it is computed, whilst in the standard Gram-Schmidt
method only the vectors \( u_k (k = i, i+1, \ldots) \) are explicitly required
to be orthogonal to \( u_i \).
PART I

In addition to being a numerically more stable method the modified method also has computational advantages in that it requires less computer storage and is very easy to program.

1.8 Sturm sequence check

Without even explaining what a Sturm sequence is, we will here state a very important sign count check that relies on a property of such sequences. For details and proof the reader is referred to BATHE and WILSON [4] and JENNINGS [3].

Consider the symmetric eigenproblem of Eq. (1.1):

\[(A - \lambda B)q = 0\]

Let

\[A_\sigma = A - \sigma B\]

Unless \(\sigma\) is exactly equal to an eigenvalue, say \(\lambda_i\), this matrix may be factorized such that

\[A_\sigma = L_{\lambda_i} D_{\lambda_i} L_{\lambda_i}^T\]

The Sturm sequence check now states that the number of negative elements in the diagonal matrix \(D_{\lambda_i}\) is exactly equal to the number of eigenvalues \((\lambda_i)\) that are (algebraically) smaller than \(\sigma\).

This is a very robust and reliable check which also applies to the special problem of Eq. (1.3); simply set \(B\) equal to \(I\) in the above equations.

PRELIMINARIES
1.9 The Krylov sequence and some of its properties

A is a square \( n \) by \( n \) symmetric matrix and \( x_0 \) is an arbitrary vector with \( n \) elements. Consider the sequence of \( m \) vectors

\[
    x_0, Ax_0, A^2x_0, \ldots, A^{m-1}x_0
\]

These vectors form a *Krylov sequence* of vectors, and the first important property of this sequence is that

*the vectors of the sequence become more and more parallel to the dominant eigenvector of matrix \( A \).*

In order to demonstrate this property let the eigenvalues of \( A \) be ordered in such a way that

\[
    \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n
\]

The corresponding eigenvectors are \( q_1, q_2, \ldots, q_n \). The arbitrary vector \( x_0 \) may be expressed as

\[
    x_0 = c_1q_1 + c_2q_2 + \ldots + c_nq_n
\]

The second vector of the sequence is

\[
    x_1 = Ax_0 = \sum_{i=1}^{n} \lambda_i c_i q_i
\]

The \((k+1)\)-th vector is

\[
    x_k = A^k x_0 = \sum_{i=1}^{n} \lambda_i^k c_i q_i
\]

Provided that \( \lambda_1 > \lambda_2 \) it follows that when \( k \) is large and \( c_j \) is non-zero

\[
    x_k = \lambda_1^k c_1 q_1
\]

If \( \lambda_1 = \lambda_2 \) we will have

\[
    x_k = \lambda_1^k (c_1 q_1 + c_2 q_2) = \lambda_1^k c_1 q_1
\]

Since \( q_1 \) and \( q_2 \) are not unique when \( \lambda_1 = \lambda_2 \), any linear combination of \( q_1 \) and \( q_2 \) is a perfectly legitimate dominant eigenvector.

The vectors of (1.58) form the columns of a so-called *Krylov matrix*

\[
    K^{(n)}(x_0) = [x_0 \ Ax_0 \ A^2x_0 \ \ldots \ A^{m-1}x_0]
\]

For \( m < n \) the columns of \( K^{(n)}(x_0) \) are linearly independent provided \( x_0 \) does not have a special relationship with \( A \).

Assume that we have constructed an orthonormal basis
PART I

\[ \mathbf{V}_j = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j] \] for which \( \mathbf{V}_j^T \mathbf{V}_j = \mathbf{I} \)

by applying for instance Gram-Schmidt orthonormalization on the columns of \( \mathbf{K}^{(0)}(\mathbf{x}_0) \) in the sequence \( \mathbf{x}_0, \mathbf{A} \mathbf{x}_0, \ldots \).

The basis \( \mathbf{V}_{j+1} \) is obtained by adding the vector \( \mathbf{v}_{j+1} \) to \( \mathbf{V}_j \), and \( \mathbf{v}_{j+1} \) is constructed by orthonormalizing the last column of \( \mathbf{K}^{(j+1)}(\mathbf{x}_0) \), that is \( \mathbf{A}^j \mathbf{x}_0 \), with respect to \( \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \). Instead of \( \mathbf{A}^j \mathbf{x}_0 \) we can use a Gram-Schmidt process on \( \mathbf{A} \mathbf{v}_j \) and write

\[ \mathbf{r}_j = \mathbf{A} \mathbf{v}_j - \sum_{i=1}^{j} \alpha_i \mathbf{v}_i \quad \text{where} \quad \alpha_i = \mathbf{v}_i^T (\mathbf{A} \mathbf{v}_j) \]

\[ \mathbf{v}_{j+1} = \mathbf{r}_j / \| \mathbf{r}_j \| \]

It can now be shown that

\[ \alpha_i = 0 \quad \text{for} \quad i < j - 1 \]

or in words:

*the vectors \( \mathbf{v}_j \) are inherently orthogonal to all previous vectors except the last two.*

This is the second important property of the Krylov sequence. It can be demonstrated as follows:

We begin with the arbitrary vector \( \mathbf{v}_1 = \mathbf{x}_0 / \| \mathbf{x}_0 \| \) and construct a new vector

\[ \mathbf{v}_2 = \mathbf{A} \mathbf{v}_1 - \alpha_1 \mathbf{v}_1 \]

such that \( \mathbf{v}_2 \) is orthogonal to \( \mathbf{v}_1 \), that is

\[ \mathbf{v}_1^T \mathbf{v}_2 = 0 \Rightarrow \mathbf{v}_1^T \mathbf{A} \mathbf{v}_1 - \alpha \mathbf{v}_1^T \mathbf{v}_1 = 0 \Rightarrow \alpha = \frac{\mathbf{v}_1^T \mathbf{A} \mathbf{v}_1}{\mathbf{v}_1^T \mathbf{v}_1} \]

Next we construct a vector

\[ \mathbf{v}_3 = \mathbf{A} \mathbf{v}_2 - \alpha_2 \mathbf{v}_2 - \beta_1 \mathbf{v}_1 \]

such that \( \mathbf{v}_3 \) is orthogonal to both \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) or

\[ \mathbf{v}_1^T \mathbf{A} \mathbf{v}_2 - \alpha_2 \mathbf{v}_1^T \mathbf{v}_2 - \beta_1 \mathbf{v}_1^T \mathbf{v}_1 = 0 \Rightarrow \beta_1 = \frac{\mathbf{v}_1^T \mathbf{A} \mathbf{v}_2}{\mathbf{v}_1^T \mathbf{v}_1} \]

and

\[ \mathbf{v}_2^T \mathbf{A} \mathbf{v}_3 - \alpha_2 \mathbf{v}_2^T \mathbf{v}_2 - \beta_1 \mathbf{v}_2^T \mathbf{v}_1 = 0 \Rightarrow \alpha_2 = \frac{\mathbf{v}_2^T \mathbf{A} \mathbf{v}_3}{\mathbf{v}_2^T \mathbf{v}_2} \]

Next we construct
THEORY

\[ \mathbf{v}_4 = \mathbf{A}\mathbf{v}_3 - \alpha_1 \mathbf{v}_3 - \beta_2 \mathbf{v}_2 - \gamma_1 \mathbf{v}_1 \]

such that \( \mathbf{v}_4 \) is orthogonal to \( \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \). Following the above procedure we find

\[ \gamma_1 = \frac{\mathbf{v}_1^T \mathbf{A} \mathbf{v}_3}{\mathbf{v}_1^T \mathbf{v}_1}, \quad \beta_2 = \frac{\mathbf{v}_2^T \mathbf{A} \mathbf{v}_3}{\mathbf{v}_2^T \mathbf{v}_2} \quad \text{and} \quad \alpha_3 = \frac{\mathbf{v}_3^T \mathbf{A} \mathbf{v}_3}{\mathbf{v}_3^T \mathbf{v}_3} \]

However, from the expression for \( \mathbf{v}_2 \) above we find that

\[ \mathbf{v}_1^T \mathbf{A} = \mathbf{v}_1^T \mathbf{A}^T = \mathbf{v}_1^T + \alpha_1 \mathbf{v}_1^T \]

Hence

\[ \gamma_1 = \frac{(\mathbf{v}_1^T + \alpha_1 \mathbf{v}_1^T) \mathbf{v}_1}{\mathbf{v}_1^T \mathbf{v}_1} = \frac{0 + 0}{\mathbf{v}_1^T \mathbf{v}_1} = 0 \]

This process goes on, and each new vector is made orthogonal to all previous vectors by expressing it in terms of only the two preceding vectors.

1.10 Change of problem - shift and inversion

Shift of origin is frequently used to improve the convergence of iterative solution methods. It is also used to investigate the occurrence of eigenvalues in a certain "range", and sometimes it can be used to enable a solution that would otherwise be impossible or very difficult to obtain by a given technique.

Consider the generalized symmetric eigenproblem of Eq. (1.1):

\[ (\mathbf{A} - \lambda \mathbf{B}) \mathbf{q} = \mathbf{0} \]

Let

\[ \lambda = \gamma + \sigma \quad \Rightarrow \quad \gamma = \lambda - \sigma \]  	(1.60)

Substitution gives

\[ (\mathbf{A} - \sigma \mathbf{B} - \gamma \mathbf{B}) \mathbf{q} = \mathbf{0} \]

or

\[ (\mathbf{A}_\sigma - \gamma \mathbf{B}) \mathbf{q} = \mathbf{0} \]  	(1.61)

where

\[ \mathbf{A}_\sigma = \mathbf{A} - \sigma \mathbf{B} \]  	(1.62)

\( \sigma \) is the shift value:
σ is often an input argument to a general purpose eigensolution subroutine or function.

Consider the original problem of Eq. (1.1) once again, that is no shift. If \( \mathbf{B} \) is a lumped \textit{(i.e.} diagonal) mass matrix in a free vibration problem with some zero diagonal elements, the problem has some \textit{infinite} eigenvalues (one for each zero mass), and a complete solution is impossible. However, the \textit{inverse} problem

\[
(\mathbf{B} - \mu \mathbf{A}) \mathbf{q} = \mathbf{0}
\]

where \( \mu = 1/\lambda \) has \( n \) finite eigenvalues (to each infinite eigenvalue of the original problem the inverse problem has a zero eigenvalue). If \( \mathbf{A} \) is positive definite Eq. (1.63) can always be solved; depending on the method of solution a shift of origin may be necessary or advantageous.

Inversion and/or shift are particularly useful techniques when complete eigensolutions are required, as is, for instance, the case for the reduced eigenproblem in the subspace iteration method.

1.11 Transformation to standard form

The solution of the special or standard symmetric eigenproblem of Eq.(1.2) is a much easier problem than the generalized problem posed by Eq.(1.1). An obvious question therefore is: can the generalized problem be transformed into an equivalent standard problem? The answer is yes, and we have several options. Mathematically we seek to replace Eq.(1.1b), that is

\[
\mathbf{A} \mathbf{q} = \lambda \mathbf{B} \mathbf{q}
\]

with the standard problem

\[
\mathbf{H} \mathbf{p} = \lambda \mathbf{p}
\]
THEORY

where $H$ is a symmetric $n$ by $n$ matrix whose eigenvectors $p_i$ are related to the eigenvectors $q_i$ in some well-defined way.

The transformation requires that we factorize $A$ or $B$ in the form

$$ A = W_A^	op W_A \quad \text{or} \quad B = W_B^	op W_B $$

(1.65)

Factorizing matrix $A$

a) Matrix $A$ is positive definite

In this case $W_A$ is the Cholesky factor, and we write the decomposition of $A$ as

$$ A = U_A^	op U_A $$

(1.66)

Substituting this expression into Eq. (1.2) gives:

$$ U_A^	op U_A q_i = \lambda_i B q_i $$

(1.67)

Premultiplication, first by $B^{-1}$ and then by $U_A^{-1}$, gives

$$ U_A B^{-1} U_A^	op U_A q_i = \lambda_i U_A q_i $$

This equation has the form of (1.64) with

$$ H = U_A B^{-1} U_A^	op $$

(1.68)

and

$$ p_i = U_A q_i $$

(1.69)

Once Eq. (1.64) has been solved with respect to $\lambda_i/p_i$, the original eigenvectors $q_i$ are recovered by use of (1.69), that is, by simple backsubstitution.

b) Matrix $A$ is positive semi-definite

The factorization (1.65) is, in special cases, possible even if the matrix is not positive definite. For instance, if $A$ is a diagonal (lumped mass) matrix with some zero diagonal entries, the factor $W_A$ exists, but it is obviously not the Cholesky factor.

Equations (1.68) and (1.69) still hold if $U_A$ is replaced by $W_A$, but since $A$ (and $W_A$) is rank deficient (singular) we have to resort to (1.67) in order to recover the original eigenvectors. Premultiplication by $B^{-1}$ and substitution of (1.69) give

$$ q_i = B^{-1} W_A p_i / \lambda_i $$

(1.70)

Transformation to standard form

23
\[ B^{-1} \] which appears in both (1.68) and (1.70) need not be formed explicitly, but we need to factorize \( B \). Since it can be indefinite, Cholesky decomposition will not always work. Instead we use \( B = L_B D_B L_B^T \) (1.71)

**Factorizing matrix \( B \)**

\( a) \ Matrix \ B \) is positive definite

In this case \( W_B \) is the Cholesky factor, and we write the decomposition of \( B \) as

\[ B = U_B^T U_B \] (1.72)

Substituting this expression into Eq. (1.1) and using \( U_B^T U_B = I \) gives:

\[ (A - \lambda, U_B^T U_B) U_B^T U_B q_i = 0 \]

Introducing

\[ p_i = U_B q_i \] (1.73)

and premultiplying with \( U_B^T \) give

\[ (H - \lambda, I) p_i = 0 \] (1.74)

where

\[ H = U_B^T A U_B^{-1} \] (1.75)

In some cases it is required to form matrix \( H \) explicitly. This is easily achieved through two forward substitution operations. Consider (1.75) and determined the auxiliary matrix

\[ G = U_B^T A \]

through forward substitution. Hence

\[ H = G U_B^{-1} = H^T = U_B^T G^T \]

is also obtained through forward substitution, and, since we know that \( H \) is symmetric, only a good half of \( H \) needs to be determined.

Once Eq.(1.74) has been solved with respect to \( \lambda, / p_i \), the original eigenvectors \( q_i \) are again recovered by simple back substitution, by use of (1.73).

\( b) \ Matrix \ B \) is positive semi-definite

Equation (1.65) into (1.1b):
THEORY

\[ Aq_i = \lambda_i W_B W_B^T q_i, \]

or, using \( W_B^T W_B^T = I \) and introducing \( \mu_i = 1/\lambda_i \),

\[ \mu_i A W_B^T W_B^T q_i = W_B W_B^T q_i. \]

Introducing

\[ p_i = W_B^T q_i \quad (1.76) \]

and premultiplication with first \( A^{-1} \) and then \( W_B^T \) give

\[ \mu_i W_B^T A^{-1} A W_B^T p_i = W_B^T A^{-1} W_B p_i, \]

or

\[ H p_i = \mu_i p_i, \]

where

\[ H = W_B^T A^{-1} W_B \quad (1.77) \]

Since there is no need to form the inverse of \( W_B \), this procedure can be used even if \( W_B \) is rank deficient. The original eigenvectors are found as

\[ \mu_i A q_i = W_B p_i \quad \Rightarrow \quad q_i = A^{-1} W_B p_i / \mu_i \quad (1.78) \]
METHODS OF SOLUTION

This chapter summarizes some of the most important computational methods used for solving the symmetric matrix eigenproblem. Most methods apply to the standard problem defined by Eq. (1.2), but some can also be used directly on the generalized problem of Eq. (1.1). This is particularly important for large, sparse matrices, since reduction from generalized to standard form will produce a much denser matrix and largely destroy properties of bandedness. This chapter will concentrate on methods for finding all eigenvalues and corresponding eigenvectors for relatively small, full matrices. The two major methods for large, sparse problems, the subspace iteration method and the truncated Lanczos method will be dealt with in a separate chapter.

Since the eigenvalues are the roots of the characteristic polynomial equation, and since there is no exact algebraic solution of polynomial equations with \( n > 4 \), the computational procedures are necessarily iterative in nature. Based on the properties they utilize, the methods fall into three categories:

1. Polynomial iteration / determinant methods;
2. Transformation methods;
3. Vector iteration methods.

This survey is by no means complete. The interested reader is referred to the basic text by Wilkinson [5] for a comprehensive treatment.
2.1 Polynomial iteration / determinant methods

The methods in this category are based on the properties of the characteristic polynomial \( p(\lambda) \) formed by expanding the determinant of the matrix

\[
S(\lambda) = A - \lambda B
\]

that is

\[
p(\lambda) = \det[S(\lambda)]
\]

The zeros of \( p(\lambda) = 0 \) determines the eigenvalues.

**Determinant search**

The basic determinant search method is a ‘brute force’ method. It is a quite general method that simply requires the evaluation of \( \det[S(\lambda)] \) for many values of \( \lambda \) and observation of the sign changes in \( \det[S(\lambda)] \). The value of the determinant is calculated by first factorizing \( S(\lambda) \) into \( L_sD_sL_s^T \).

In its basic form the method has some serious deficiencies. Many factorizations may be required to locate an eigenvalue, although convergence may be speeded up by applying some form of interpolation. Furthermore, no indication of the position of the computed root in the eigenvalue spectrum can be given, and closely or coincident (multiple) eigenvalues may be missed altogether.

**Sign count methods**

All practical methods of this category rely heavily on the Sturm sequence property stated in Section 1.8 (page 18).

Again the basic numerical operation is the factorization of \( S(\lambda) \) into \( L_sD_sL_s^T \). However, instead of computing the determinant of \( S(\lambda) \), the number of negative elements in the diagonal matrix \( D_s \) determines the number of eigenvalues that are (algebraically) smaller than \( \lambda \). By use of some bracketing technique, e.g. bisection, the interval in which a particular eigenvalue must lie can be limited, and the eigenvalue can thus be determined to any desired level of accuracy.

These methods can be used to solve the generalized problem in Eq. (1.1) directly, but since a large number of factorizations are involved, the techniques are competitive only if a small number of eigenvalues are required for matrices with small bandwidths. The methods only find the eigenvalues. If the corresponding eigenvectors are also required they must be found by some other techniques, such as inverse iteration (see below).
2.2 Transformation methods

A transformation method transforms the matrix under investigation into another matrix with the same eigenvalues. Usually many such transformations are carried out before the eigenvalues can be obtained either by inspection (the matrix is transformed into diagonal form) or the matrix is in a form which can be easily analyzed by alternative procedures.

The most general transformation which retains the eigenvalues of a matrix is the similarity transformation, see Section 1.4. Since this transformation applies to an individual matrix, we present the methods of this category in terms of a symmetric matrix $H$. For the special eigenproblem $H$ is identical to matrix $A$ of Eq. (1.2), whereas for the generalized problem $H$ is an equivalent matrix obtained as described in Section 1.11.

With reference to Eq. (1.21), the grand strategy is:

To nudge $H$ towards diagonal form by a sequence of similarity transformations:

$$H \Rightarrow P_1^\dagger HP_1 \Rightarrow P_2^\dagger (P_1^\dagger HP_1) P_2 \Rightarrow \ldots$$

There are two rather different sets of techniques for implementing the grand strategy:

1. Construct individual $P_i$'s as explicit ‘atomic’ transforms designed to perform the task
   a) all the way to diagonal form (Jacobi), or
   b) most of the way, to tridiagonal or Hessenberg form (Givens, Householder).

2. Factorization methods:
   Factorize $H$ into a left ($F_L$) and a right ($F_R$) factor:
   $$H = F_L F_R$$
   or
   $$F_L^\dagger H = F_R$$
   Multiplying the factors in reverse order is equivalent to a similarity transformation (see also Section 1.4 on page 10):
   $$F_R F_L = F_L^\dagger H F_L$$
   Rutishauser’s LR method, Francis’ QR method and the slightly modified QL method all fall in this group.

**Jacobi diagonalization**

Each Jacobi transformation eliminates one pair of off-diagonal elements in a symmetric matrix $H$. 

Transformation methods
The orthogonal transformation matrix is a rotation matrix of the form
\[
P = \begin{bmatrix}
1 & c & -s \\
-s & c & p \\
p & q & n
\end{bmatrix}
\]
where
\[c = \cos \alpha, \quad s = \sin \alpha\] (2.3)

The 'angle' \(\alpha\) is determined so as to make \(H_{pq}\) and \(H_{qp}\) equal to zero.

The elements which have been eliminated do not necessarily stay zero, and hence the method is iterative in character.

If the matrix after \(k-1\) transformations is designated \(H^{(k)}\) then the \(k\)-th transformation may be written as
\[
H^{(k+1)} = P^T H^{(k)} P_k
\] (2.4)

and the eigenvector matrices of \(H^{(k)}\) and \(H^{(k+1)}\) are related by
\[
Q^{(k+1)} = P^T Q^{(k)}
\] (2.5)

If a total of \(s\) transformations are necessary to diagonalize the matrix we have
\[
Q^{(s+1)} = P_2^T P_{s-1}^T \ldots P_2^T P_1^T Q^{(1)}
\]

However, since \(Q^{(s+1)}\) is the matrix of eigenvectors of a diagonal matrix, we have
\[
Q^{(s+1)} = I
\]
and the eigenvectors of the original matrix \(H^{(1)}\) appear as columns of
\[
Q^{(1)} = P_1 P_2 \ldots P_{s-1} P_s
\] (2.6)

The basic Jacobi procedure applies to the standard eigenproblem. It is a simple, robust and reliable procedure, but, as we shall see in PART II, it is not the most efficient method. It has, however, the advantage of providing \(n\) orthogonal eigenvectors, also in the case of multiple eigenvalues.
A generalized Jacobi procedure, due to Falk and Langemeyer [6], is available for the generalized eigenproblem defined by Eq. (1.1). Matrices $A$ and $B$ are reduced towards diagonal form simultaneously using a non-orthogonal generalization of the Jacobi transformation. The procedure, which requires $B$ to be positive definite, is often used in connection with subspace iteration (more about this in Chapter 3), and it works quite well in spite of the fact that a formal proof of its convergence does not exist, see Parlett [1]. Vissing [10] claims that the algorithm also works for matrices with zero entries on the diagonal.

Givens tridiagonalization

Givens' method adopts the Jacobi transformation to produce a tridiagonal matrix with the same eigenvalues as the starting matrix [7]. The process is non-iterative and more efficient than Jacobi diagonalization. But it requires the resulting tridiagonal matrix to be analyzed separately. The procedure consists of a series of transformations representing rotations in planes $(p,q)$ in the order $(2,3), (2,4), \ldots, (2,n)$ followed by $(3,4), (3,5), \ldots, (3,n)$ etc. Any particular rotation $(p,q)$ will knock out the two elements in position $(p-1,q)$ and $(q,p-1)$. Eigenvectors $q_j$ can only be determined after the eigensolution $(\lambda, u)$ of the tridiagonal matrix has been accomplished. Then

$$q_s = P_s P_{s-1} \ldots P_2 P_1 q_1$$

where $s = (n-1)(n-2)/2$ signifies the last Givens transformation.

Householder tridiagonalization

Householder's method also reduces a symmetric matrix to a similar tridiagonal form, but it is computationally more efficient than the basic Givens' method. The transformation matrix is a reflection matrix of the form

$$P = I - 2ww^T$$

where $w$ is a (column) vector whose Euclidean norm is unity, i.e.,

$$w^T w = 1$$

The transformation matrix is orthogonal since

$$PP^T = (I - 2ww^T)(I - 2ww^T) = I$$

and, because it is also symmetric,
The appropriate elements of an entire column (and row) are elimi­nated in one transformation

The LR transformation

Rutishauser’s LR method uses a similarity transformation involving the triangular factors of a matrix to produce another matrix of greater diagonal dominance.

If this transformation is applied iteratively, the diagonal elements normally converge to the eigenvalues of the original matrix.

It is not an efficient method for fully populated matrices. However, when the matrix has certain patterns of zero elements (tridiagonal, Hessenberg) these are retained by the transformation, thus improving the efficiency.

If $H^{(k)}$ is the matrix obtained from the $(k-1)$-th transformation, its triangular decomposition mat be written as

$$H^{(k)} = L_k R_k \Rightarrow R_k = L_k^{-1} H^{(k)}$$  \hspace{1cm} (2.9)

where $L_k$ is a lower triangular matrix and $R_k$ is an upper triangular matrix. Multiplying the factors in reverse order completes the transformation, i.e.

$$H^{(k+1)} = R_k L_k = L_k^{-1} H^{(k)} L_k$$  \hspace{1cm} (2.10)

For a symmetric positive definite matrix, the use of its Cholesky factors will ensure that symmetry is maintained in the sequence of transformed matrices.

The rate of convergence can be improved by using shifts.

The QR and QL transformations

The QR transformation, due to FRANCIS [9], is similar to the LR transformation except that the first factor is replaced by an orthog­onal matrix. Thus if

$$H^{(k)} = Q_k R_k \Rightarrow R_k = Q_k^{-1} H^{(k)}$$  \hspace{1cm} (2.11)

where $Q_k$ is an orthogonal matrix ($Q_k^{-1} = Q_k^T$) and $R_k$ is an upper triangular matrix. Hence

$$H^{(k+1)} = R_k Q_k = Q_k^{-1} H^{(k)} Q_k$$  \hspace{1cm} (2.12)

and the eigenvalues of $H^{(k+1)}$ are the same as those of $H^{(k)}$. Since $Q_k$ is orthogonal it follows from (2.11) that

METHODS OF SOLUTION
where $Q_i^T$ is the product of orthogonal matrices of Givens or Householder type. If applied to a full matrix $H$, the Givens triangular factorization in (2.13) will result in an operational cost of every factorization being $O(n^3)$. Since many such factorizations are performed during an iteration cycle, it is generally not feasible to carry out the QR transformation on a full matrix. Instead, if $H$ is first reduced to the similar tridiagonal form $T$, for instance by the method of Householder, then the cost is dramatically reduced. In general, there will now be $n-1$ orthogonal matrices $P_i$, each eliminating one subdiagonal element, giving

$$P_{n-1}^T \ldots P_1^T H^{(k)} = R_k$$

and

$$H^{(k+1)} = R_k P_1 P_2 \ldots P_{n-1}$$

as the operative equations for one iteration.

Furthermore, if the initial matrix $H^{(1)}$ is tridiagonal, the subsequent transformations will always give a tridiagonal $H^{(k)}$.

The main advantage of the QR method is that any shift strategy can be adopted without incurring either the need to interchange rows or the likelihood of loss of accuracy due to small pivots. When a shift is applied, the algorithm changes to

$$(H^{(k)} - \sigma_i I) = Q_i R_k \Rightarrow H^{(k+1)} = R_k Q_i + \sigma_i I$$

(2.14)

According to Wilkinson [5] the best shift value is found as the eigenvalue of the submatrix formed by the last two rows and columns of $H^{(k)}$. This gives cubic convergence.

Another useful characteristic of the method is that, as $k$ increases, the eigenvalues appear in ascending order from the bottom right corner of the diagonal of $H^{(k)}$. Thus as each new eigenvalue appears the matrix $H^{(k)}$ can be reduced in size by omitting its last row and column, and the iterations are continued on a matrix of reduced order.

The eigenvectors can be found by a series of matrix multiplications, in a way similar to what was used by Jacobi's method. However, this is quite a costly method, and a much faster method is to use inverse iteration on $H^{(1)}$. This is particularly efficient if $H^{(1)}$ is a tridiagonal matrix.
PART I

QR is more robust and reliable than LR, and it is also more efficient. The last factor may equally well be a lower triangular matrix, and QR may be replaced QL.

2.3 Vector iteration methods

All methods of this category are based on one of two very important properties of the Krylov sequence, see Section 1.9 (page 19). The methods described in this section make use of the first property. The method utilizing the second property, the Lanczos method, is dealt with in Chapter 3.

Power iteration

Consider the standard eigenproblem of Eq. (1.2) and assume an arbitrary (random) start vector \( \mathbf{x}_0 \). By repeatedly multiplying this vector by matrix \( \mathbf{A} \) we obtain, as shown in Section 1.9, a vector that is more and more parallel with the dominant eigenvector, that is the eigenvector corresponding to the largest eigenvalue.

Since an eigenvector can be arbitrarily scaled, it is convenient to normalize the trial vector after each premultiplication. An iterative algorithm to determine \( q_1 \) may therefore be expressed by the two equations

\[
\mathbf{z}_k = \mathbf{A} \mathbf{x}_k \tag{2.15}
\]

\[
\mathbf{x}_{k+1} = \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|} \tag{2.16}
\]

The best eigenvalue estimate corresponding to the approximate eigenvector \( \mathbf{x}_k \) is determined as the Rayleigh quotient (see Section 1.5 on page 11) given by

\[
\lambda_1^{(k)} = \frac{\mathbf{x}_k^T \mathbf{A} \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{x}_k} = \frac{\mathbf{x}_k^T \mathbf{z}_k}{\mathbf{x}_k^T \mathbf{x}_k} \equiv \rho(\mathbf{x}_k)
\]

Once

\[
\frac{\lambda_1^{(k)} - \lambda_1^{(k-1)}}{\lambda_1^{(k)}} < \text{tolerance}
\]

then

\[
\lambda_1 = \lambda_1^{(k)} \quad \text{and} \quad (\mathbf{q}_1 = \mathbf{z}_k/\|\mathbf{z}_k\|).
\]

Once \( \lambda_1 \) and \( q_1 \) have been found it is possible to find the next eigenpair by the same procedure provided the trial vector is made orthogonal to \( q_1 \) throughout the iteration. This normally requires a
THEORY

Gram-Schmidt ‘purification’ in each iteration cycle. Shifting is another way to make the process converge towards another eigenpair.

Inverse iteration

For most physical problems we are more interested in the lowest eigenpair. This is the dominant eigenpair of the inverse problem

\[
(A^{-1} - \frac{1}{\lambda} I)q = 0
\]

or

\[
(A^{-1} - \bar{\lambda} I)\bar{q} = 0
\]

Power iteration with \(A^{-1}\) gives the dominant eigenpair \((\bar{\lambda}, \bar{q})\) of \(A^{-1}\) which is the lowest (or least dominant) eigenpair \((\lambda_n, q_n)\) of \(A\).

The inverse matrix \(A^{-1}\) need not be determined explicitly since, if triangular decomposition \(A = LDL^T\) is performed, the two operations

\[
Ly_k = x_k \quad \Rightarrow \quad y_k = L^{-1}x_k \quad \text{(forward substitution)}
\]

and

\[
DL^Tz_k = y_k \quad \Rightarrow \quad z_k = L^{-T}D^{-1}y_k \quad \text{(backsubstitution)}
\]

are equivalent to the premultiplication

\[
z_k = A^{-1}x_k
\]

Both the standard (forward) power iteration and the inverse iteration can be applied to the generalized eigenproblem of Eq. (1.1). Assuming a ‘start’ vector \(x_1\) we have

\[
Ax_k = Bx_k \quad \Rightarrow \quad z_k
\]

and

\[
x_{k+1} = \frac{z_k}{(z_k^T B z_k)^{1/2}} \quad \text{(2.19)}
\]

The normalization of (2.19) will ensure that the trial vector remains \(B\)-orthonormal, that is

\[
x_{k+1}^T B x_{k+1} = 1
\]

In an actual computer implementation it is more effective to iterate as follows:

Vector iteration methods
PART I

Assuming $\mathbf{y}_1 = \mathbf{Bx}_1$ we evaluate for $k = 1, 2, \ldots$

$$\mathbf{A} \mathbf{z}_k = \mathbf{y}_k \Rightarrow \mathbf{z}_k$$

$$\mathbf{r}_{k+1} = \mathbf{B} \mathbf{z}_k$$

$$\rho(\mathbf{z}_k) = \frac{\mathbf{z}_k^T \mathbf{y}_k}{\mathbf{z}_k^T \mathbf{r}_{k+1}}$$

$$\mathbf{y}_{k+1} = \frac{\mathbf{r}_{k+1}}{(\mathbf{z}_k^T \mathbf{r}_{k+1})^{1/2}}$$

Provided $\mathbf{y}_1 \mathbf{q}_1 \neq 0$ we have

$$\mathbf{y}_{k+1} \Rightarrow \mathbf{B} \mathbf{q}_1$$ and $\rho(\mathbf{z}_k) \Rightarrow \lambda_1$ as $k \Rightarrow \infty$.

It is also possible to iterate with several vectors simultaneously. Of the various algorithms proposed, the most important is the so-called subspace iteration method which is described in some detail in the next chapter.
This chapter is devoted to the two main methods used for partial solution of large and sparse generalized, linear and symmetric eigenvalue problems in structural analysis, the \textit{subspace iteration method} and the \textit{truncated Lanczos method}. The presentation owes much to SEHMI [11].

Both methods are developed in some detail, including comments on computer implementation, but numerical examples are left for Part II.

### 3.1 Subspace iteration

This is the most popular version of the \textit{simultaneous inverse vector iteration} methods. The method, which is generally applied to the generalized problem of Eq. (1.1), seeks to determine all the $m$ lowest (or least dominant) eigenpairs through an inverse iteration scheme that operates simultaneously on $p$ trial vectors - or a $p$-dimensional subspace - where $p \geq m$.

Full advantage may be taken of the symmetry and the sparseness of the matrices, and the method is based on a combination of the power method and the Rayleigh-Ritz procedure.

#### The basic algorithm

Consider the eigenproblem of Eq. (1.1), or rather the alternative formulation of Eq. (1.9), that is

$$ AQ = BQA $$

(3.1)
where \( \mathbf{Q} \) and \( \mathbf{A} \) contain some or all eigenvectors and corresponding eigenvalues, respectively. The eigenvectors are normalized (see Section 1.3) such that

\[
\mathbf{Q}^\top \mathbf{B} \mathbf{Q} = \mathbf{I} \quad (3.2)
\]

and

\[
\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \mathbf{A} \quad (3.3)
\]

The \( m \) lowest eigenpairs \((\lambda_i, \mathbf{q}_i), i = 1, 2, \ldots, m\), of (3.1) are determined through a certain number of iteration cycles, starting with \( p \) (\( \geq m \)) linearly independent trial vectors

\[
\mathbf{V}_1 = \left[ \mathbf{v}_1^\top \mathbf{v}_2^\top \ldots \mathbf{v}_p^\top \right]
\]

Having obtained improved vectors \( \mathbf{V}_k \), the next iteration may be expressed as follows:

From

\[
\mathbf{A} \mathbf{W}_{k+1} = \mathbf{B} \mathbf{V}_k \quad (3.4)
\]

solve for

\[
\mathbf{W}_{k+1} = \mathbf{A}^{-1} \mathbf{B} \mathbf{V}_k \quad (3.5)
\]

Determine the reduced ('projected') subspace matrices

\[
\overline{\mathbf{A}}_{k+1} = \mathbf{W}_{k+1}^\top \mathbf{A} \mathbf{W}_{k+1} = \mathbf{W}_{k+1}^\top \mathbf{B} \mathbf{V}_k \quad (3.6)
\]

and

\[
\overline{\mathbf{B}}_{k+1} = \mathbf{W}_{k+1}^\top \mathbf{B} \mathbf{W}_{k+1} \quad (3.7)
\]

Solve the reduced \((p \times p)\) eigenproblem

\[
\overline{\mathbf{A}}_{k+1} \mathbf{x}_{k+1} = \overline{\mathbf{B}}_{k+1} \mathbf{x}_{k+1} \quad (3.8)
\]

for \( \overline{\mathbf{A}}_{k+1}, \overline{\mathbf{B}}_{k+1} \) such that

\[
\mathbf{x}_{k+1}^\top \overline{\mathbf{B}}_{k+1} \mathbf{x}_{k+1} = 1 \quad (3.9)
\]

and

\[
\mathbf{x}_{k+1}^\top \overline{\mathbf{A}}_{k+1} \mathbf{x}_{k+1} = \overline{\mathbf{A}}_{k+1} \quad (3.10)
\]

Improved approximations to the eigenvectors are now found as

\[
\mathbf{v}_{k+1} = \mathbf{W}_{k+1} \mathbf{x}_{k+1} \quad (3.11)
\]

and as \( k \) increases then

\[
\overline{\mathbf{A}}_{k+1} \rightarrow \mathbf{A}_p \quad \text{and} \quad \mathbf{v}_{k+1} \rightarrow \mathbf{Q}_p
\]

Where \( \Lambda_p = [\lambda_1 \lambda_2 \ldots \lambda_p] \) and \( \mathbf{Q}_p = [\mathbf{q}_1 \mathbf{q}_2 \ldots \mathbf{q}_p] \). Here it is assumed that \( \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_p \).
Equation (3.4) is an inverse iteration step with a matrix instead of a vector. Each time it is performed the step is equivalent to the pre-multiplication of \( V_k \) by \( A - B \). Providing the first trial vectors (the 'start vectors') are not orthogonal to the least dominant eigenvector \( (q_1) \), this, as shown in Section 2.3, has the effect of making all vectors in \( W_{k+1} \) becoming increasingly more parallel, and converging towards \( q_1 \). To prevent this, some form of vector 'purification' or orthogonalization of \( W_{k+1} \) is needed before it is used in the next iteration.

In subspace iteration this is achieved by solving the reduced (subspace) eigenproblem of (3.8). This is essentially an application of the Rayleigh-Ritz procedure and is equivalent to simultaneous minimization of \( \rho \) Rayleigh quotients, see Section 1.6 (page 13). The postmultiplication of \( W_{k+1} \) by the eigenvectors \( X_{k+1} \) in (3.11) represents the 'purification' process that leaves the eigenvector approximations \( V_{k+1} \) orthogonal with respect to \( B \). This is easily seen by applying Eqs. (3.11), (3.7) and (3.9):

\[
V_{k+1}^T B V_{k+1} = X_{k+1}^T W_{k+1}^T B W_{k+1} X_{k+1} = I
\]

(3.12)

Hence \( V_{k+1} \) is \( B \)-orthonormal (even if \( V_k \) is not!).

Also, in view of (3.11), (3.6) and (3.10),

\[
V_{k+1}^T A V_{k+1} = X_{k+1}^T W_{k+1}^T A W_{k+1} X_{k+1} = \Omega_{k+1}
\]

(3.13)

It should be emphasized that for (3.12) and (3.13) to hold, it is necessary to solve (3.8) such that (3.9) and (3.10) are satisfied.

The iteration process described above may also be used to find the \( m \) eigenvalues and corresponding eigenvectors that are closest to a specified shift value \( \sigma \), see Section 1.10 (page 21). The iteration would then be carried out for the modified problem

\[
(A_\sigma - \gamma B)q = 0
\]

(3.14)

or

\[
A_\sigma Q = B Q \Gamma
\]

(3.15)

where

\[
A_\sigma = A - \sigma B \quad \text{and} \quad \gamma = \lambda - \sigma \Rightarrow \lambda = \gamma + \sigma.
\]

**Convergence**

Iteration is carried out until the relative error is less than a specified error tolerance value (\( \varepsilon \)) for each of the \( m \) lowest eigenvalues, that is until
PART I

\[
\frac{\omega_i^{(k+1)} - \omega_i^{(k)}}{|\omega_i^{(k)}|} \leq \varepsilon \quad \text{for} \quad i = 1, 2, \ldots, m \quad (3.16)
\]

The convergence is fastest for the lowest eigenvalues, and it can be shown \([4, 11]\) that any particular \(v_i\) converging towards \(q_i\) (where \(i \leq p\)) will have an error proportional to \((\lambda_i/\lambda_{p+1})^k\) after the \(k\)th iteration. Since the eigenvalues are effectively calculated by the Rayleigh quotient, the error in the eigenvalue \(\omega_i\) will be proportional to \((\lambda_i/\lambda_{p+1})^{2k}\) after \(k\) iterations. This follows from Rayleigh's principle, see Section 1.5 (page 11). Still, if \(\lambda_{p+1}\) is close to \(\lambda_p\), the value of \(k\) will have to be large in order to secure convergence. For practical computations it is therefore essential that the number of iteration vectors \((p)\) is higher than the number of required eigenpairs \((m)\). A much quoted value for \(p\), first presented by Bathe and Wilson \([4]\), is

\[
p = \min \{2m, m + 8\} \quad (3.17)
\]

This is certainly not an optimal value for all cases, as will be shown in PART II.

The eigenvalues computed by subspace iteration are essentially Ritz values, and therefore, as pointed out in Section 1.6 (page 13), they represent upper bounds to the true eigenvalues, that is, \(\lambda_i \leq \omega_i\).

The standard subspace iteration method described here is well suited for finding a moderate number of eigenpairs for large problems, but, as will be demonstrated in PART II, it becomes less and less efficient as the value of \(m\) increases. This problem can to some extent be overcome by applying some shifting strategies and extract the eigenpairs in groups. Shifts may also speed up convergence, but time saved due to this speedup may easily be spent in expensive matrix factorizations required to safeguard (through sign counts) against missing some eigenpairs. A general purpose ('black box') eigensolver making active use of shift strategies will be hard to code, and its efficiency will probably be even more problem dependent than the standard procedure.

Start vectors

The number of subspace iterations needed for convergence clearly depends on how closely the initial trial or starting vectors \(V_1\) span the least dominant \(p\)-dimensional subspace. If \(V_1\) contained exact linear combinations of just \(q_1, q_2, \ldots, q_p\), the process would converge to the exact eigenvectors in one cycle.
THEORY

In practice, one can in some cases make reasonable guesses about the lowest or perhaps a few of the lowest eigenvectors (or modes of free vibration or buckling), but in the general case it is not only difficult to make good guesses, it may also be a cumbersome process to transfer the guesses to the eigensolver routine. For a general purpose eigensolver routine we clearly need some automatic scheme or schemes for generating the starting vectors $V_1$. However, in order to cater for those few cases where the user has some prior knowledge of the eigenvectors, the eigensolver routine should also accept user supplied start vectors.

The start vectors should at least

- be linearly independent, and
- they should not be deficient of components of (or orthogonal to) any of the $p$ first eigenvectors $q_i$.

The most obvious choice of start vectors is a set of randomly or pseudo-randomly generated vectors. It is very unlikely that such a set would not meet the two basic requirements. On the other hand, such a set obviously does not contain any information about the eigenvectors of the specific problem.

Bathe and Wilson [4] propose a selection procedure for $V_1$ based on practical engineering experience with problems of vibration. The first column of $V_1$ is taken to be the diagonal of the mass matrix (B), thus ensuring that all degrees of freedom associated with mass are excited in the fundamental mode. The second column of $V_1$ is taken as a unit vector $e_j$ with an entry of $+1$ placed at the position corresponding to the degree of freedom having the largest ratio $B_{jj}/A_{jj}$, where $A_{jj}$ and $B_{jj}$ are the $(i,j)$ elements on the leading diagonal of the stiffness (A) and mass (B) matrix, respectively. The third column of $V_1$ is taken to be the unit vector $e_k$ where $k$ corresponds to the second largest ratio $B_{kk}/A_{kk}$, and so on.

Other, more sophisticated procedures have been suggested for the generation of start vectors, but none of these seem to be significantly superior over a wide range of problems.

In order to ensure that components of all eigenvectors are present in the set of trial vectors, a much used technique is to feed in a random or pseudo-random vector, in the position of the last trial vector, after each iteration cycle.

Computational procedure

The computational procedure implemented in subroutine MSSIT used in PART II may be outlined as follows:

1. Select start vectors - store in $V$. 

Subspace iteration
PART I

2. If necessary, modify matrix $A$:

$$A_a = A - \sigma B$$

and factorize $A / A_a$ into $L_{A_a}D_{A_a}L_{A_a}^T$.

**NOTE:** The factors $L_{A_a}, D_{A_a}$ may also be input information.


4. Start of iteration loop: set $V = W$ ($= BV_k$)

5. Solve for the Ritz basis vectors ($W_{k+1}$): $V \leftarrow A^{-1}V$

6. Determine the subspace matrices $\tilde{A}$ and $\tilde{B}$:

$$\tilde{A} = V^TAV = V^TW$$

This follows from the fact that

$$AV = (AW_{k+1} = BV_k) = W$$

The multiplication $V^TW$ should take advantage of the symmetry of $\tilde{A}$. Next compute

$$W = BV \ (= BW_{k+1})$$

and finally

$$\tilde{B} = V^TW \ (= W_{k+1}^TBW_{k+1})$$

7. Solve reduced eigenproblem

$$\tilde{A}X = \tilde{B}X\Omega$$

We shall return to this problem in the next section.

8. If convergence:

   determine $B$-orthonormal eigenvectors

   $$V \leftarrow VX \ (= V_{k+1})$$

   and exit loop,

else

   determine new Ritz-vectors premultiplied by $B$:

   $$W \leftarrow WX \ (= BW_{k+1}X_{k+1})$$

   and go to step 4.

The matrix or array storage requirement of the iteration procedure outlined above is, in addition to $A$ and $B$: two $n \times p$ matrices ($V$ and $W$) plus an auxiliary $n \times 1$ vector, three $p \times p$ matrices ($\tilde{A}$, $\tilde{B}$ and $X$), two $p \times 1$ vectors and three auxiliary $p \times 1$ vectors.
**Solution of the reduced eigenproblem**

The solution of the reduced, subspace eigenproblem in step 7 of the computational procedure represents a crucial part of a general purpose subspace iteration eigensolver. The process requires a complete solution to this problem which we restate as follows:

\[(G - \mu_i H)x_i = 0\]

Most methods available for the complete solution of this problem require matrix \(H\) to be positive definite.

If matrix \(B\) is positive definite (as it will be for a consistent mass matrix in a dynamic problem) we have:

\[H = B_{k+1}, \quad G = A_{k+1} \text{ and } \mu_i = \omega_i\]

and there should be no problems, since \(H\) will also be positive definite.

If \(A\) is positive definite and \(B\) is not (which may well be the case in a linearized buckling problem or in the case of a lumped mass matrix in a dynamic problem) we proceed as follows:

1. The shift value is zero (\(\sigma = 0\)) or \(A_\sigma = A - \sigma B\) is positive definite (can be established by a sign count of \(D_{A_\sigma}\)). In this case we proceed with:

\[H = A_{k+1}, \quad G = B_{k+1} \text{ and } \mu_i = 1/\omega_i\]

and again we would not expect any problems.

2. \(\sigma \neq 0\), and \(A_\sigma\) is indefinite. In this case we may have a problem. Our best chance is to try to make \(\tilde{A}\) positive definite. We define:

\[\tilde{\sigma} = (1 + \alpha)\sigma\]

and form

\[\tilde{A}_\sigma = \tilde{A} + \tilde{\sigma} B\]

which is assumed to be positive definite, and we set:

\[H = \tilde{A}_\sigma, \quad G = \tilde{B} \text{ and } \mu_i = 1/(\omega + \tilde{\sigma})\]

So what about \(\alpha\)? If \(B\) is semi-definite (as a lumped mass matrix would be) \(\alpha \geq 0\) should render \(\tilde{A}_\sigma\) positive definite as long as \(A\) is positive definite (which we assumed at the start). However, if \(B\) is indefinite it is difficult to argue for a non-zero value of \(\alpha\). Since it is not desirable to make \(\alpha\) a user-specified input argument to our routine, \(\alpha = 0\) seems to be our best bet. However, there is no guarantee that this will work, and for this special case of problems (which is hard to imagine in a real world structural problem) one
PART I

cannot exclude failure.

Speeding up the process

It follows from the discussion about convergence, and it is confirmed by examining the behavior in actual computations, that the subspace iteration method accept the eigenpairs from below. Or in other words, the lowest eigenvalues are the first to converge. As pointed out by Seimi [11] the number of operations during an iteration cycle can be reduced by ‘freezing’ already converged eigenvectors, and by not recomputing these eigenvectors, the process can be speeded up.

With reference to the numbered steps in the computational procedure, modifications will have to be made in steps 4, 5, 6 and 8. Great care should be exercised in the implementation of these modifications as mistakes are easily made (this author spent more time than he cares to think about debugging this sequence).

A direct operations count shows that the reduction in computational effort can be quite significant. However, some of this reduction could be lost in slower convergence, since, by freezing eigenvectors as soon as they pass the tolerance criteria, the error still contained in these vectors will remain and may ‘contaminate’ the following eigenvectors sufficiently to slow down their convergence. If left in the process, as indicated by the unmodified computational procedure described on page 42, already converged eigenvectors will continue to ‘improve’, and this ought to benefit the convergence of the remaining eigenpairs. This aspect will be investigated in PART II.

3.2 Truncated Lanczos

The method of LANCZOS [12] was first presented in 1950 as a procedure for transforming a given $n \times n$ matrix $H$ into a similar tridiagonal matrix $T$ in $n$ iteration steps. This is achieved by an orthogonal similarity transformation such that

$$T = V^* H V_n$$ where $V^T V_n = I$ (3.18)

$V_n = [v_1 \ v_2 \ldots \ v_n]$ are referred to as the Lanczos vectors, and they are generated from one single trial vector by use of the second property of the Krylov sequence described in Section 1.9 (page 19). As proposed by LANCZOS the algorithm is, however, numerically unstable due to a loss of orthogonality in the Lanczos vectors. Introducing reorthogonalization, for instance by the Gram-Schmidt procedure, resulted in more computational work than Householder’s
method, and Lanczos' method disappeared from the scene for about twenty years.

In 1970 Ojalvo and Newman [13] applied the algorithm with reorthogonalization to solve for the lower natural frequencies and mode shapes of large structural dynamic problems. Their approach was to truncate the tridiagonalization process after \( p \) iterations (where \( p \) is much smaller than \( n \)) in order to produce a reduced tridiagonal matrix \( T_p \) of order \( p \times p \). An eigensolution of this matrix provided very good approximations to about \( p/2 \) of the dominant eigenvalues of the original matrix \( H \). This truncated version proved to be extremely efficient and highly competitive with other methods in finding the dominant eigenpairs of \( H \).

Another important work at about the same time is that of Paige [14] who conducted a rigorous error analysis of the Lanczos algorithm. One of Paige's many findings was that, properly implemented, the Lanczos algorithm could still obtain accurate results without reorthogonalization, although this was at the expense of generating a tridiagonal matrix \( T \) whose dimension could be significantly larger than that of \( H \). This inevitably lead to multiple copies of the required eigenvalues being obtained from eigensolution of \( T \). Which of these belonged to \( H \), particularly if the original problem contained exact multiple roots?

These works sparked off a renewed interest in the Lanczos method, and through much work and many papers the method is now developed into perhaps the most efficient and robust (?) technique for finding a limited number of eigenvalues/eigenvectors of large symmetric eigenproblems. The work has been directed in two major directions: 1) reorthogonalization of the Lanczos vectors as they are computed, and 2) the Paige-style approach without reorthogonalization. We shall limit ourselves to the first approach, and since reorthogonalization is fairly expensive, a main concern is to make this operation as efficient as possible. Another important issue in the quest for a robust, general purpose Lanczos eigensolver is to establish a reliable stop criteria, ensuring that all requested eigenvalues/eigenvectors are found with the least possible computational effort.

The Lanczos algorithm

The presentation in this section of the basic Lanczos algorithm, which applies to the special eigenproblem, follows in all major aspects the one given by Sefi [11].

Consider the special, symmetric eigenproblem

\[
Hy = \theta y \quad \text{or} \quad HY = Y\Theta
\]

(3.19)

where \( H \) is a symmetric \( n \times n \) matrix. The eigenvectors are orthogonal, that is...
PART I

\[ Y^TY = YY^T = I \]  
(3.20)

Hence

\[ Y^THY = \Theta = [\theta_1 \ \theta_2 \ldots \ \theta_n] \]  
(3.21)

The Lanczos method uses the second property of the Krylov sequence, described in Section 1.9 (page 19), to construct a set of \( n \) orthonormal Lanczos vectors \( \mathbf{v}_j \) or

\[ \mathbf{V}_n = [\mathbf{v}_1 \ \mathbf{v}_2 \ldots \mathbf{v}_n] \quad \mathbf{V}_n^T\mathbf{V}_n = I \]  
(3.22)

such that

\[ \mathbf{V}_n^T\mathbf{H}\mathbf{V}_n = \mathbf{T}_n \]  
(3.23)

where \( \mathbf{T}_n \) is a tridiagonal matrix similar to \( \mathbf{H} \) (in the sense that it has the same eigenvalues). If we introduce

\[ \mathbf{Y} = \mathbf{V}_n\mathbf{Z}_n \]  
(3.24)

Eq. (3.19) can be written as

\[ \mathbf{T}_n\mathbf{Z}_n = \mathbf{Z}_n\Theta \]  
(3.25)

From (3.23) it also follows that

\[ \mathbf{H}\mathbf{V}_n = \mathbf{V}_n\mathbf{T}_n \]  
(3.26)

Hence

Equating vectors:

\[ \mathbf{H}\mathbf{v}_1 = \alpha_1\mathbf{v}_1 + \beta_2\mathbf{v}_2 \]
\[ \mathbf{H}\mathbf{v}_2 = \beta_2\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \beta_3\mathbf{v}_3 \]
\[ \vdots \]

\[ \mathbf{H}\mathbf{v}_i = \beta_i\mathbf{v}_{i-1} + \alpha_i\mathbf{v}_i + \beta_{i+1}\mathbf{v}_{i+1} \]  
(3.27)
THEORY

\[ H \mathbf{v}_n = \beta_n \mathbf{v}_{n-1} + \alpha_n \mathbf{v}_n \]

Equation (3.27) is a three-term recurrence formula, the basis of which is the second property of a Krylov sequence demonstrated in Section 1.9. In conjunction with the orthogonality relationship, \( \mathbf{v}_i^T \mathbf{v}_j = \delta_{ij} \), it is used to obtain all the \( \alpha_i, \beta_i \), and \( \mathbf{v}_i \).

The sequence is started from an arbitrary non-null vector \( \mathbf{v}_1 \) satisfying

\[ \mathbf{v}_1^T \mathbf{v}_1 = 1 \]

From (3.27):

\[ \mathbf{v}_1^T H \mathbf{v}_1 = \alpha_1 \mathbf{v}_1^T \mathbf{v}_1 + \beta_2 \mathbf{v}_1^T \mathbf{v}_2 \implies \alpha_1 = \mathbf{v}_1^T H \mathbf{v}_1 - \beta_2 \mathbf{v}_2 = r_2 \]

Also

\[ \beta_2 = \|r_2\| \quad \text{and} \quad \mathbf{v}_2 = r_2 / \beta_2 \]

See also the derivations in Section 1.9.

The algorithm can be formally written as follows:

Let \( \mathbf{v}_0 = 0 \)

\( r_1 \) is an arbitrary start vector, and

\( \beta_1 = \|r_1\| \)

Then for \( i = 1, 2, \ldots, n \) do:

\[ \mathbf{v}_i = r_i / \beta_i \]  \hfill (3.28a)

\[ \mathbf{u}_i = H \mathbf{v}_i \]  \hfill (3.28b)

\[ \alpha_i = \mathbf{v}_i^T \mathbf{u}_i \]  \hfill (3.28c)

\[ r_{i+1} = \mathbf{u}_i - \beta_i \mathbf{v}_{i-1} - \alpha_i \mathbf{v}_i \]  \hfill (3.28d)

\[ \beta_{i+1} = \|r_{i+1}\| = (r_{i+1}^T r_{i+1})^{1/2} \]  \hfill (3.28e)

end-do.

A slightly different and (according to Paige) more stable algorithm is as follows:

Let \( \mathbf{v}_0 = 0 \)

\( r_1 \) is an arbitrary start vector, and

\( \beta_1 = \|r_1\| \)

Then for \( i = 1, 2, \ldots, n \) do:

\[ \mathbf{v}_1 = \frac{r_1}{\beta_1} \]  \hfill (3.29a)

\[ \mathbf{u}_1 = H \mathbf{v}_1 \]  \hfill (3.29b)

\[ \alpha_1 = \mathbf{v}_1^T \mathbf{u}_1 \]  \hfill (3.29c)

\[ r_{i+1} = \mathbf{u}_i - \beta_i \mathbf{v}_{i-1} - \alpha_i \mathbf{v}_i \]  \hfill (3.29d)

\[ \beta_{i+1} = \|r_{i+1}\| = (r_{i+1}^T r_{i+1})^{1/2} \]  \hfill (3.29e)
The difference is the subtraction of the term $\beta_i v_{i-1}$.

Algorithm (3.28) gives

$$v_i = r_i/\beta_i$$

(3.29a)

$$u_i = Hv_i - \beta_i v_{i-1}$$

(3.29b)

$$\alpha_i = v_i^T u_i$$

(3.29c)

$$r_{i+1} = u_i - \alpha_i v_i$$

(3.29d)

$$\beta_{i+1} = \|r_{i+1}\| = (r_{i+1}^T r_{i+1})^{1/2}$$

(3.29e)

end-do.

The difference is the subtraction of the term $\beta_i v_{i-1}$.

Algorithm (3.28) gives

$$r_{i+1} = Hv_i - \beta_i v_{i-1} - \alpha_i v_i$$

where $\alpha_i = v_i^T u_i = v_i^T Hv_i$.

Algorithm (3.29) also gives

$$r_{i+1} = Hv_i - \beta_i v_{i-1} - \alpha_i v_i$$

where $\alpha_i = v_i^T (Hv_i - \beta_i v_{i-1}) = v_i^T Hv_i$ since $v_i^T v_{i-1} = 0$.

Hence the two algorithms are theoretically equivalent.

If, in exact arithmetic, the start vector $r_1$ is not deficient in any of the eigenvectors $y_i$, the process will generate exactly $n$ orthonormal Lanczos vectors. If $\beta_i$ becomes zero it is possible to restart the process, with a vector that is orthonormal to all previous vectors, and thus obtain a block tridiagonal matrix $T_n$.

In real arithmetic the chances of a ‘breakdown’ (indicated by $\beta_i=0$) are very small. A far more serious consequence of real (‘finite digit’) arithmetic is the gradual loss of orthogonality in the Lanczos vectors, and in practice some form of reorthogonalization is necessary. More about this later.

We note that the matrix $H$ is not altered during the process, and it is only used to form the matrix-vector product $Hv_i$.

In practice complete tridiagonalization by (3.28) or (3.29) is rarely performed - Householder’s method would be better. Instead a truncated process, leading to a tridiagonal $p \times p$ matrix, $T_p$, where $p < n$ and usually $p \ll n$, is used, that is:

$$V_p = [v_1 \ v_2 \ \cdots \ v_p] \quad V_p^T V_p = I \quad (3.30)$$

$$V_p^T HV_p = T_p \quad (3.31)$$

The smaller eigenproblem
provides very good estimates of about \( p/2 \) of the dominant eigenpairs \((\theta_i, y_i)\) of (3.19). The reason for this becomes clearer when the process is shown to be equivalent to an application of the Rayleigh-Ritz technique.

In Lanczos' method, \( p \) assumed 'displacement functions' of the form \( Y_p = [y_1, y_2, \ldots, y_p] \) are taken to be linear combinations of the Lanczos vectors \( V_p \), so that

\[
Y_p = V_p Z_p
\]  

(3.33)

where \( Z_p = [z_1, z_2, \ldots, z_p] \) are the Ritz coordinates to be determined. The Lanczos vectors are the Ritz basis vectors. Forming the \( pxp \) generalized Rayleigh quotient gives

\[
\rho(Y_p) = \frac{Z_p^T V_p H V_p Z_p}{Z_p^T V_p V_p Z_p} = \frac{Z_p^T T Z_p}{Z_p^T Z_p}
\]  

(3.34)

whose \( p \) stationary values are given by the eigensolution

\[
T_p Z_p = Z_p \Psi
\]  

(3.35)

where

\[
\rho(Y_p) = \Psi = [\psi_1, \psi_2, \ldots, \psi_p]
\]

\[\psi_i \geq \psi_2 \geq \ldots\]

The eigenvectors \( Z_p \), which satisfy \( Z_p^T Z_p = 1 \) define the Ritz coordinates which, when combined with the \( V_p \), will give the Ritz vectors \( Y_p \) in (3.33), and the corresponding Ritz values \( \Psi \) are the 'optimum' eigenvalue estimates obtainable from the space spanned by the Lanczos vectors \( V_p \).

The way in which the Lanczos vectors are formed, through continuous multiplication with \( H \) (as in the power method), explains the fact that the method produces eigenvalue estimates from the dominant end of the spectrum.

The eigenproblem of (3.35) can be solved by any of a number of methods, e.g., QR and inverse iteration or an implicit QL method. The most common criteria used for signalling convergence is that the Ritz values calculated at consecutive stages satisfy the condition

\[
|\psi_i^{(t+1)} - \psi_i^{(t)}|/|\psi_i^{(t+1)}| \leq \varepsilon
\]  

(3.36)

where \( \varepsilon \) is a predefined error tolerance.
Reorthogonalization

It can be shown (Paige) that loss of orthogonality is initiated when a Ritz value converges.

Hence the strength of the method (that approximately $p/2$ of the eigenvalues of $T_p$ are good approximations to the $p/2$ dominant eigenvalues of $H$) is also the cause of its most serious problem: loss of orthogonality. Since the process will magnify any rounding errors present, it is of vital importance to monitor the loss of orthogonality and to reorthogonalize a new Lanczos vector to the previous ones whenever necessary.

However, the rounding errors will normally bring in sufficient ‘new’ information to prevent breakdown of the process. The success of the algorithm hinges on balancing the need for maintaining orthonormality with the need for bringing in sufficient numerical ‘noise’ to prevent breakdown or the loss of certain vectors.

a) Full reorthogonalization

Various methods are available, but here only modified Gram-Schmidt is considered. In order to include this extra reorthogonalization step, the algorithm (3.29) need to be altered.

Let $\tilde{r}_{i+1}$ be the vector produced in (3.29d), instead of $r_{i+1}$, which need to be reorthogonalized. The orthogonalized vector is then given as

$$r_{i+1} = \tilde{r}_{i+1} - \sum_{j=1}^{i} v_j (\tilde{r}_{i+1} v_j)$$

(3.37)

This step should be inserted between (3.29d) and (3.29e), with step (3.29d) being changed to read

$$\tilde{r}_{i+1} = u_i - \alpha_i v_i$$

(3.38)

b) Partial reorthogonalization

Ideally, what is required is a procedure which will predict when orthogonality is lost so that the corrective measure of performing a Gram-Schmidt orthogonalization is applied only when it is necessary.

We assume that orthogonality is satisfied at step $i$, that is

$$V_i^T V_i = I$$

At the next step the orthogonality relation becomes:
The key here is the vector 

\[ \mathbf{v}_i^T \mathbf{v}_{i+1} \]

Each element of this vector is the dot product of \( n \)-dimensional vectors. Calculating these dot products is in itself a costly operation and requires nearly as many operations as those needed for the full reorthogonalization procedure of (3.37).

Fortunately, the dot products \( \mathbf{v}_i^T \mathbf{v}_{i+1} \), and hence the loss of orthogonality, can be predicted by a recursion formula which may be derived by some clever manipulation of the original Lanczos three-term recurrence relation (3.27):

\[ \mathbf{v}_{i+1} \beta_{i+1} = \mathbf{H} \mathbf{v}_i - \alpha_i \mathbf{v}_i - \mathbf{v}_{i-1} \beta_i \]

In block matrix form this is

\[
\begin{pmatrix}
\mathbf{H} & \mathbf{V}_i \\
\mathbf{V}_i^T & \mathbf{T}_i
\end{pmatrix} =
\begin{pmatrix}
\mathbf{V}_i \\
\mathbf{T}_i
\end{pmatrix} + 0
\]

Postmultiplying (3.39) with \( \mathbf{e}_i \) and premultiplying with \( \mathbf{V}_i^T \) give

\[ \beta_{i+1} \mathbf{V}_i^T \mathbf{v}_{i+1} = \mathbf{V}_i^T \mathbf{H} \mathbf{v}_i - \mathbf{v}_i^T \mathbf{T}_i \mathbf{e}_i \]

Since \( \mathbf{H} = \mathbf{H}^T \) and \( \mathbf{T}_i = \mathbf{T}_i^T \), we have from (3.39)

\[ \mathbf{V}_i^T \mathbf{H} = \mathbf{T}_i \mathbf{V}_i^T + \beta_{i+1} \mathbf{e}_i \mathbf{v}_{i+1} \]

This into (3.40) gives

\[ \beta_{i+1} \mathbf{V}_i^T \mathbf{v}_{i+1} = (\mathbf{T}_i \mathbf{V}_i^T + \beta_{i+1} \mathbf{e}_i \mathbf{v}_{i+1}) \mathbf{e}_i - \mathbf{V}_i^T \mathbf{T}_i \mathbf{e}_i \]

and since

\[ \mathbf{V}_i \mathbf{e}_i = \mathbf{v}_i \quad \text{and} \quad \mathbf{V}_i \mathbf{T}_i \mathbf{e}_i = \beta_i \mathbf{v}_{i-1} + \alpha_i \mathbf{v}_i \]

we get

Truncated Lanczos

51
which is the desired recurrence formula. It relates the dot products \( V^T_i v_{i+1} - e_i v^T_i v_i \) to \( V^T_i v_i \) and \( V^T_i v_{i-1} \). According to SHEMI [11] this formula was known to PAIGE [14], but it was first applied by SIMON [15].

If full reorthogonalization is used, Gram-Schmidt produces vectors for which

\[
V_i^T v_j = O(\text{eps}) = 0 \quad i \neq j
\]

where \( \text{eps} \) designate the machine precision, that is \( \text{eps} \) is the smallest number attainable on the computer such that

\[
1 + (\text{eps}) > 1
\]

In other words, the Gram-Schmidt orthogonalization is assumed to leave the vectors orthogonal to full machine precision. In actual fact this is seldom the case with only a ‘single-pass’ orthogonalization (where each new vector is orthogonalized to all previous vectors once). Repeating the orthogonality process once (‘double-pass’) or twice (‘three-pass’) will normally improve the orthogonality slightly, although the difference between a ‘double-pass’ and a ‘three-pass’ process is hardly noticeable.

SHEMI [11] refers to PARLETT and SIMON who suggest that if the vectors \( V_i \) are orthogonal to the level

\[
V_i^T v_j = O[\sqrt{\text{eps}}] \approx 0 \quad i \neq j
\]

all benefits of full reorthogonalization will be maintained. This suggests a threshold value of

\[
\varepsilon_i = \sqrt{\text{eps}}
\]

for the partial reorthogonalization procedure. If the estimated dot products between a new vector and its predecessors exceeds this threshold reorthogonalization is called for. If this takes place for Lanczos vector number \( i \), both PARLETT and SIMON recommend strongly that reorthogonalization is also carried out for the next vector, that is for vector number \( i+1 \). Their arguments for this consecutive application of the Gram-Schmidt process are also quoted by SHEMI [11].

c) A simple ‘half’ reorthogonalization process

Considering the fundamental basis for the Lanczos method, the fact that a new Lanczos vector is inherently orthogonal to all previous vectors except the last two (a direct consequence of a Krylov
sequence property), it is tempting to propose the following very simple, 'ad hoc' procedure:

The first two Lanczos vectors are formed without any reorthogonalization (vector number two is formed to be explicitly orthonormal to the first one, via the $\alpha$ and $\beta$ factors). The next two vectors are formed and explicitly orthogonalized, by Gram-Schmidt, to the previous vectors. Then we form two vectors without using explicit (Gram-Schmidt) orthogonalization to the previously generated vectors. And so we continue to form two vectors with explicit reorthogonalization and two without, ending up with approximately half the computational effort of a full ('single-pass') reorthogonalization implementation.

This simple method actually works quite well, as will be demonstrated by numerical examples in Part II, but it should be emphasized that its basis is more an 'inspiration' than a well founded argument.

**Starting vector**

In exact arithmetic the procedure will break down - indicated by a zero $\beta$, being computed - if the start vector $r_1$ is deficient in (orthogonal to) any of the dominant eigenvectors. If this happens, which in real arithmetic is very unlikely due to rounding errors, it is, as already mentioned, quite possible to restart the process by choosing a new (random) vector and make it orthogonal to the already generated Lanczos vectors.

The process therefore does not seem to be very sensitive to the choice of starting vector. Nevertheless, the speed by which the process finds the required eigenvalues seems to be best served by a starting vector containing as much information as possible about as many of the dominant eigenvectors as possible. This suggests a random or pseudo-random start vector. Other suggestions that have been made include using the diagonal of matrix $B$ as a starting vector, or a start vector of unit elements. The 'preconditioning' of the start vector by premultiplication with $H$ has also been advocated.

For a general purpose eigensolver routine a user provided start vector should also be an option. However, whichever start vector is used, it should be normalized to unit length before it is being introduced into the algorithm.

**Eigenvalue extraction and stop criteria**

A crucial question when implementing a general purpose eigensolver based on the truncated Lanczos method is when to stop the process? Or, in other words, how and when can we be sure of really
Part I

having found all the requested eigenvalues? This is particularly important for problems with multiple eigenvalues, as for instance in 3D problems with double symmetry.

Assuming that we have requested $m$ of the dominant eigenvalues of $H$, eigenvalue extraction, that is solution of the reduced problem of Eq. (3.32), is carried out after each Lanczos step marked by a circle in the diagram below:

The first solution of (3.32) takes place for $k = m$, the next for $k = \text{int}\left\{3xm/2\right\}$ and then for each $s$’th step.

The process is halted when at least $m$ ‘good’ eigenvalues have been found or after the maximum number of steps, $p$, have been carried out. The ‘good’ eigenvalues are those eigenvalues that have not only passed the tolerance test of Eq. (3.36) in two consecutive eigenvalue extractions, but have also maintained their position in the ordered eigenvalue list during these two extractions. Parameters $m$, $p$, $s$ and $\varepsilon$ are all input parameters to the eigensolver routine.

This cut-off procedure, which may lead to some over-kill for values of $s$ greater than 1, is motivated by the desire to detect and pick up multiple eigenvalues. ‘Acceptable’ eigenvalues are those that have passed the tolerance test after a particular eigenvalue extraction. These are almost certainly true eigenvalues, but they may not all have their proper position in an ordered eigenvalue sequence, since multiples of ‘earlier’ values may not yet have appeared on the scene, or they may be preceded by some yet unacceptable eigenvalues.

Parameters $p$, $\varepsilon$ and $s$ will be subject to numerical testing in Part II.

Application to the generalized eigenproblem

So far we have been concerned with finding the dominant eigenvalues of the special problem stated by Eq. (3.19). However, the problem we really want to solve can be stated as follows:

Determine the $m$ eigenvalues $\lambda_i$ and (optionally) the corresponding eigenvectors $q_i$ of the generalized, symmetric eigenproblem of Eq. (3.1), that is

$$Aq_i = Bq_i \lambda_i$$

or

LARGE MATRIX EIGENSOLUTIONS
THEORY

\[(A - \lambda B)q_i = 0\]

that are closest to a specified shift value \(\sigma\).

\[p(\lambda) = \det(A - \lambda B)\]

Introducing

\[\gamma = \lambda - \sigma \Rightarrow \lambda = \gamma + \sigma\]

we may formulate the problem as

\[(A_\sigma - \gamma B)q = 0\] (3.42a)

or

\[A_\sigma Q = BQ\gamma^{-1}\] (3.42b)

where

\[A_\sigma = A - \sigma B\] (3.43)

The task may now be defined as: determine the \(m\) least dominant eigenvalues/eigenvectors of the generalized eigenproblem of (3.42). In other words, we seek the \(m\) numerically smallest eigenvalues of (3.42). This is equivalent to finding the \(m\) dominant (numerically largest) eigenvalues of the inverse problem:

\[(B - \theta A_\sigma)q_i = 0\] (3.44)

where

\[\theta_i = \frac{1}{\gamma_i}\] (3.45)

A truncated Lanczos tridiagonalization of a symmetric matrix \((H)\) may now be used to determine the dominant eigenvalues of this matrix. Since we seek the least dominant eigenvalues \((\lambda_i)\) we need to work with Eq. (3.44) and use Lanczos to determine \(\theta_i\).

The first step in this process is to transform the generalized problem of (3.44) to the special problem of (3.19), that is to

\[Hy_i = \theta_i y_i\] or \[HY = Y\Theta\]

where \(H\) is a symmetric matrix and the eigenvector \(y_i\) is related to \(q_i\).

In order to achieve this we need to factorize \(B\) or \(A_\sigma\) into
PART I

\[ B = U_B^T U_B \]  

(3.46)

or

\[ A_\sigma = U_{A_\sigma}^T U_{A_\sigma} \]  

(3.47)

respectively, where \( U_B \) and \( U_{A_\sigma} \) are the Cholesky factors of \( B \) and \( A_\sigma \), respectively; they are upper triangular, or possibly diagonal matrices. Since we can only determine the Cholesky factor for a positive definite matrix (or for a semi-definite diagonal matrix with non-negative terms), a general eigensolver should offer both factorizations and leave it to the discretion of the user to choose which is the most appropriate for his/her problem.

The transformation to the special form is accomplished as described in Section 1.11 (page 22). However, since the matrices \( A \) and \( B \) have changed place in the equation, we repeat the basic steps here.

**a) Factorizing matrix \( A_\sigma \)**

Equation (3.47) into (3.44) and using \( U_{A_\sigma}^T U_{A_\sigma} = I \):

\[ (B - \theta \, U_{A_\sigma}^T U_{A_\sigma}) U_{A_\sigma} U_{A_\sigma} q_i = 0 \]  

(3.48)

Introducing

\[ y_i = U_{A_\sigma} q_i \]  

(3.49)

and premultiplying with \( U_{A_\sigma}^T \) give

\[ (H - \theta I) y_i = 0 \]  

(3.50)

where

\[ H = U_{A_\sigma}^T B U_{A_\sigma} \]  

(3.51)

Once (3.50) has been solved with respect to \( \theta / y_i \), the original eigenvectors \( q_i \) are recovered by backsubstitution, see (3.49),

\[ q_i = U_{A_\sigma}^T y_i \]  

(3.52)

**b) Factorizing matrix \( B \)**

Equation (3.46) into (3.44):

\[ U_B^T U_B q_i = \theta A_\sigma q_i \]  

(3.53)

Premultiplication, first by \( A_\sigma^{-1} \) and then by \( U_B \), give

1. We continue to use the notation \( A_\sigma \), even if this alternative in most cases will only be used for \( \sigma = 0 \).
THEORY

\[ U_n A_\sigma^{-1} U_n^T U_n q_i = \theta_i U_n q_i \]

or

\[ H y_i = \theta_i y_i \]

where

\[ H = U_n A_\sigma^{-1} U_n^T \]  \hspace{1cm} (3.54)

and

\[ y_i = U_n q_i \]  \hspace{1cm} (3.55)

If \( B \) is a full rank matrix, that is \( U_B \) is the true Cholesky factor, \( q_i \) is found from (3.55) by backsubstitution. If, on the other hand, \( B \) is rank deficient (e.g., a diagonal matrix with some diagonal terms equal to zero, as is normally the case for a lumped mass matrix), we have to resort to Eq. (3.53). Premultiplication by \( A_\sigma^{-1} \) and substitution of (3.55) give

\[ q_i = A_\sigma^{-1} U_n^T y_i / \theta_i \]  \hspace{1cm} (3.56)

\( A_\sigma^{-1} \) which appears in both (3.54) and (3.56) need not be formed explicitly, but we need to factorize \( A_\sigma \), and since it may be indefinite we cannot use Cholesky. Instead we use the factorization of (1.11), that is

\[ A_\sigma = L_{\lambda_\alpha} D_{\alpha} L_{\lambda_\alpha}^T \]  \hspace{1cm} (3.57)

Some computational aspects

In the implementation of the algorithm, matrix \( H \) is only involved in step (3.29b) where the product \( Hv_i \) needs to be formed.

Depending on which matrix is factorized this operation is performed as follows:

a) Matrix \( A_\alpha \) is factorized:

\[ Hv_i = U_{A_\alpha}^T B U_{A_\alpha}^{-1} v_i \]

Let \( f \) and \( g \) be temporary \( n \times l \) vectors. Then

1. \( U_{A_\alpha} f = v_i \Rightarrow f = U_{A_\alpha} v_i \) (backsubstitution)
2. \( g = B f \) (multiplication)
3. \( U_{A_\alpha}^T (H v_i) = g \Rightarrow Hv_i = U_{A_\alpha}^T g \) (forward substitution)

b) Matrix \( B \) is factorized:

\[ Hv_i = U_B A_\sigma^{-1} U_B^T v_i \]

Let again \( f \) and \( g \) be temporary \( n \times l \) vectors. Then
PART I

1. \( f = U^T \mathbf{v}_i \) (multiplication)

2. \( A_\alpha \mathbf{g} = f \Rightarrow \mathbf{g} = A_\alpha^{-1} f \) (solution)

3. \( H \mathbf{v}_i = U_\beta \mathbf{g} \) (multiplication)

Step 2 involves both forward and backward substitution once \( A_\alpha \) is factorized, according to (3.57). The factorization is performed once, whereas the substitutions are carried out for each Lanczos step. It should be noted that the multiplications in step 1 and 3 involves triangular matrices \((U_\beta)^T\), and the total computational effort in these two steps is therefore about the same as it is for the one multiplication step in the case of \( A \) being factorized, since this involves a full matrix \((B)\).

However, this alternative requires both matrices to be factorized.

Determining the eigenvectors

Once the \( m \) dominant eigenvalues \( \psi_i \) of Eq. (3.35) has been accepted as the \( m \) dominant eigenvalues \( \theta_i \) of Eq. (3.19) we may determine the corresponding eigenvectors \( Q_m \) of the original problem stated by Eq. (3.1).

Consider first Eq. (3.35):

This equation is solved such that

\[
Z_p^T Z_p = I_p \quad \text{and} \quad Z_p^T Z_p^T = I_m
\]  \hspace{1cm} (3.58)

From (3.24)

\[
Y_m = V_p Z_p^T
\]  \hspace{1cm} (3.59)

If \( H = U_\lambda^T B U_\lambda^{-1} \), then from (3.52)

\[
Q_m = U_\lambda^{-1} Y_m = U_\lambda^{-1} V_p Z_p^T
\]  \hspace{1cm} (3.60)

and
\[ Q_m B Q_m = (Z^p_m)^T V_p^T U_p^T B U_p^{-1} V_p Z^m_p \]
\[ = (Z^p_m)^T H V Z^m_p \]
\[ = (Z^p_m)^T T p Z^m_p = (Z^p_m)^T \Psi m \]
\[ = \Psi m = [\psi_1, \psi_2, \ldots, \psi_m] \]

Hence the eigenvectors \( Z^m_p \) obtained from (3.35) must be normalized such that

\[ (Z^m_p)^T T p Z^m_p = 1 \]
or

\[ (Z^m_p)^T Z^m_p = (\Psi m)^{-1} \quad (3.61) \]

before \( Q_m \) is obtained from (3.60). If \( \Psi m \) contains zero or negative values we need to perform a shift before we scale - a shift does not affect the eigenvectors. Hence if \( \psi_k \) is the algebraically smallest element of \( \Psi m \), and if

\[ \psi_k < \varepsilon, \quad \text{then} \quad s = |\psi_k| + \varepsilon, \]

and

\[ z_i \leftarrow \frac{1}{\sqrt{\psi_i + s}} z_i \quad (3.62) \]

A suitable value for \( \varepsilon \) is \( (eps)^{1/4} \), where \( eps \) is the machine precision, see page 52.

If \( H = U_B A_o^{-1} U_B^T \) and \( B \) is not rank deficient, then from (3.55)

\[ Q_m = U_B^{-1} Y_m = U_B^{-1} V_p Z^m_p \quad (3.63) \]

and

\[ Q_m^T B Q_m = (Z^p_m)^T V_p^T U_B^{-1} Y_m (U_B^T U_B)^{-1} U_B^{-1} V_p Z^m_p = 1 \]

Hence no need for scaling.

If \( H = U_B A_o^{-1} U_B^T \) and \( B \) is rank deficient, then from (3.56)

\[ Q_m = A_o^{-1} U_B Y_m (\Psi m)^{-1} = A_o^{-1} U_B V_p Z^m_p (\Psi m)^{-1} \quad (3.64) \]

and

\[ Q_m^T B Q_m = (\Psi m)^{-1} Y_m U_B A_o^{-1} U_B^T U_B A_o^{-1} U_B^{-1} Y_m (\Psi m)^{-1} \]
\[ = (\Psi m)^{-1} Y_m H H Y_m (\Psi m)^{-1} \]
PART I

Substituting

\[ HY_m = Y_m \Psi_m \]

we find

\[ Q^T B Q_m = Y^T_m Y_m = (Z^T_p) V_p V_p Z_p = 1 \]

and scaling is not needed.
PART II
Numerical results
This chapter outlines the purpose and scope of the numerical tests carried out, and it describes in some detail the free vibration problem used in the tests. The computer platform, both hardware and software, is also briefly discussed.

4.1 Purpose and scope of the numerical tests

Over a period of some three decades the author has been involved with development of library type software, in the form of Fortran subroutines, for numerical (finite element type) computations of structural problems. From the early eighties this work has focused on the development of an in-house library, called the SAM library [17]. This library consists of a series of more or less self-contained 'packages' of subroutines, each package devoted to the solution of a specific problem or class of problems.

One of these packages is the EIG package which is concerned with the symmetric, matrix eigenproblem. It contains routines for both the special and the generalized problem, and most methods described in Part I have been implemented. Hence, more than one subroutine is available for a given problem:

- For the special eigenproblem three subroutines/algorithms are available, all of which assume a full \((n \times n)\) matrix as input.
- For the generalized problem two subroutines are available for finding all eigenvalues and (optionally) all eigenvectors for full \((n \times n)\) matrices.
PART 11

- For large sparse matrices, subroutines are available for finding some eigenvalues and (optionally) the corresponding eigenvectors for the generalized eigenproblem by two methods (subspace iteration and truncated Lanczos). Subroutines are available for two different storage formats: the so-called 'skyline' or profile storage scheme and a 'true' sparse storage scheme.

Some of the subroutines are 'old' and quite well tested, others are the results of more recent developments and their performance has not previously been demonstrated.

Being library routines most of them have been implemented with alternative solution strategies and/or optional capabilities, and the user will have to specify which alternative/option he or she will use in a particular application. This sometimes leaves the user to resolve questions that are not easy and straightforward.

The main purpose of the numerical computations reported in the following chapters is, besides validation (demonstrate correctness), to demonstrate the accuracy, robustness and above all the relative efficiency of the different methods/subroutines. Hopefully the potential user of these or similar routines will also find some answers or guidelines to his or her questions concerning the 'best' values for some key parameters.

The results are presented and discussed in three separate chapters, one for each of the three classes of problems identified above. The emphasis is on the last class of problem, namely the large, sparse generalized eigenproblem. And within this class, the truncated Lanczos method receives most attention. In addition to the comparison between alternative methods/subroutines it is also shown how the total computational effort is distributed amongst a handful of well-defined computational tasks, for different types and sizes of problems.

4.2 Computational efficiency and accuracy

Efficiency

Computational efficiency is primarily concerned with computational effort, measured in terms of the number of certain arithmetic operations or the computertime, the so-called CPU-time\(^1\), required for the solution of a given task. However, the amount of computer storage required is also an important aspect of computational efficiency.

\(^1\) Time spent in central processing unit.
It can easily be argued that an operation count, which gives the number of arithmetical operations needed to carry out a particular sequence of matrix manipulations, is a more objective measure of computational effort than the actual (measured) CPU-time of the same sequence, since the latter involves a number of implementation dependent issues, not least the skill of the programmer. However, from the user’s point of view, the most relevant yardstick for computational efficiency is undoubtedly the CPU-time, and indirectly also the storage requirement. It should also be kept in mind that the combination of modern hardware (with several levels of ‘memory speed’ and perhaps more than one processor) and optimizing compilers may produce other CPU-times than suggested by the operation count for given (and often complex) computational tasks.

In this study it is the relative efficiency, between different methods and between different sequences within the same subroutine, that is of interest. Furthermore, the skill of the programmer is of minor importance since this is primarily a one man project, and large portions of the computations, in most subroutines, are carried out by the same (low level) functions (such as the dot product function).

On the basis of the above discussion, and also for convenience, computational efficiency is, in this report, based on measured CPU-times. For some of the subroutines only the total CPU-time is measured, whereas the main subroutines for the large sparse eigenproblems are also ‘instrumented’ with internal time measurements. In the latter case CPU-time is measured (and accumulated) for certain key operations, thus enabling the total CPU-time to be broken down into times required for well defined numerical operations, such as matrix-vector multiplication, reorthogonalization etc.

It should be noted that CPU-times measured by monitoring the computer’s ‘CPU-clock’, through a series of references to installation dependent routines, can vary quite significantly. This has to do with such factors as the load on the computer (the total number of applications running at the time of computation) and the resolution on the clock which is normally of the order of 0.01 second. Accumulation of very small elapsed times may therefore, due to the resolution, lead to inaccuracies. As an example the following 8 CPU-times (in seconds) were recorded as the total CPU-times solving the same eigenproblem (with LANCZ2) 8 times, one after the other, on an otherwise ‘empty’ computer (PC running no other applications):

70.2  71.3  69.1  69.0  68.7  68.6  68.5  69.1

Here is a variation of about 4 per cent.

All computations reported here have been carried out on PCs (see Section 4.4 below) running no other applications.
PART II

Accuracy

For the eigenvalues the accuracy is, for all iterative procedures, given in terms of the relative error, see for instance Eq. (3.16).

For the eigenvectors, however, it is more difficult to formulate such a measure. A good indication, if not an absolute measure, is obtained by monitoring the deviation of the matrix products $Q^TQ$ (special problem) and $Q^TBQ$ (generalized problem) from the identity matrix. For this purpose a special routine, that will find the (numerically) largest and smallest diagonal and off-diagonal terms of a square matrix, was developed. A typical output from this subroutine is:

SOME CHARACTERISTICS OF THE MATRIX
eigenvectors - transpose * mass matrix * eigenvectors :

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 20)

<table>
<thead>
<tr>
<th>ELEMENT</th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000015E+00</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999983E-01</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>4.771241724193381E-15</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>6.183340514956392E-20</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

4.3 Description of test problem

For the generalized eigenproblem all results reported in this study are obtained through free, undamped vibration analyses of a regular 3D frame structure. This problem is mathematically defined by

$$(K - \omega^2M)x = 0$$ (4.1)

where $K$ is the stiffness matrix, $M$ is the mass matrix and $\omega$ is the frequency of vibration, and where the displacement components of $x$ constitute the shape of the vibration (mode shape).

For the special eigenproblems of Chapter 5 the stiffness matrix $K$ of the test frame is used.

The test frame is shown in Figure 4.1. It is a regular 3D frame where all columns are fixed at the base. All column and beam elements are assumed to be rigidly connected at the nodes. The bay widths in the x- and y-directions, $a_x$ and $b_y$, respectively, are constant, and so is the story height $h_z$. The number of bays in the two directions, $M$ and $N$, as well as the number of stories, $P$, may be varied independently, thus producing problems with different matrix
Figure 4.1 Test frame
characteristics in terms of both size and position of non-zero elements.

All elements in all test frames are circular tubes with

external diameter = 500 mm and wall thickness = 30 mm

with the following material properties:

\[ E = 2.1 \times 10^{11} \text{ N/mm}^2 \quad G = 8.3 \times 10^{10} \text{ N/mm}^2 \quad \rho = 20000 \text{ kg/m}^3. \]

For all frames: \( a_x = h_z = 3.0 \text{ m} \).

For all symmetric frames: \( b_y = 3.0 \text{ m} \).

For most (all but one) non-symmetric frames: \( b_y = 5.0 \text{ m} \).

The only mass considered is the distributed mass of the tubular frame members. Three mass models are available in the test program: lumped, 'diagonalized' and consistent mass models. Only the first, resulting in a semi-definite diagonal mass matrix \( M \), and the last, resulting in a positive definite mass matrix with the same (identical) storage properties as the stiffness matrix \( K \), are considered.

The types of frames considered are listed in Table 4.1

### Table 4.1: Test frame types

<table>
<thead>
<tr>
<th>Label</th>
<th>Symmetry</th>
<th>Mass</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No</td>
<td>Consistent</td>
<td>Variable</td>
</tr>
<tr>
<td>B</td>
<td>No</td>
<td>Lumped</td>
<td>Variable</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
<td>Consistent</td>
<td>Stocky</td>
</tr>
<tr>
<td>D</td>
<td>No</td>
<td>Consistent</td>
<td>Medium</td>
</tr>
<tr>
<td>E</td>
<td>No</td>
<td>Consistent</td>
<td>Slender</td>
</tr>
<tr>
<td>F</td>
<td>Yes</td>
<td>Consistent</td>
<td>Stocky</td>
</tr>
<tr>
<td>G</td>
<td>Yes</td>
<td>Consistent</td>
<td>Slender</td>
</tr>
<tr>
<td>H</td>
<td>Almost</td>
<td>Consistent</td>
<td>Slender</td>
</tr>
</tbody>
</table>

The various frame types are recognized by a capital letter, from A to H.

Within each of the four defined frame shapes, frames of different size are defined and recognized by a number, according to Table 4.2. The resulting number of equations (NEQ) is also included for each combination of shape and size. It should be noted that \( n (= \text{NEQ}) \) only includes the free, unconstrained degrees of freedom - all specified degrees of freedom are eliminated from the matrices.
Table 4.2: Test frame shapes and sizes

<table>
<thead>
<tr>
<th>Shape</th>
<th>Number</th>
<th>$M$</th>
<th>$N$</th>
<th>$P$</th>
<th>NEQ (no. of eqns.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>540</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>960</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>10</td>
<td>1500</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>2160</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>2940</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td>3840</td>
</tr>
<tr>
<td>Slender</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>540</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>30</td>
<td>1620</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>50</td>
<td>2700</td>
</tr>
<tr>
<td>H</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>70</td>
<td>3780</td>
</tr>
<tr>
<td>Medium</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>25</td>
<td>1200</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>25</td>
<td>2100</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>25</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>12</td>
<td>1</td>
<td>25</td>
<td>3900</td>
</tr>
<tr>
<td>Stocky</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>384</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>1296</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>3072</td>
</tr>
<tr>
<td>F</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>6000</td>
</tr>
</tbody>
</table>

A particular test frame is identified by a type label and a size number, that is by *label-number*, for instance A-4, which designates a non-symmetric frame of variable shape (4 by 4 by 10) with a consistent mass model and comprising 1500 equations.

It should be emphasized that for all free vibration problems (in Chapters 6 and 7), the eigenvalues presented are the values of $\lambda = \omega^2$. In other words, the numbers given are the squares of the frequencies, not the frequencies themselves.
4.4 Computer and programming aspects

Hardware

All numerical results are obtained by different types of PC:

- **Computer I**
  - Compaq desktop
  - 90 MHz Pentium processor
  - 36 Mb RAM
  - Windows NT 4.0

- **Computer II**
  - Compaq laptop (Armada 1550T)
  - 130 MHz Pentium processor
  - 80 Mb RAM
  - Windows NT 4.0

- **Computer III**
  - Desktop PC
  - 130 MHz Pentium processor
  - 64 Mb RAM
  - Windows 95

- **Computer IV**
  - Desktop PC
  - 200 MHz / MMX Pentium processor
  - 64 Mb RAM
  - Windows 95

Software

All subroutines and functions are written in standard Fortran 77, and all floating point data are specified, stored and manipulated in ‘double precision’. This means that all numerical operations are performed with a precision level of about 16 decimal digits.

For Computer I results are obtained by a program compiled with Microsoft’s Fortran PowerStation 4.0, whereas Digital’s Visual Fortran 5.0 was used to generate the program used on Computers II, III and IV. The latter compiler is now replacing the first which is no longer supported by Microsoft. This should be of little if any consequence, since results obtained by different computers are never directly compared. It seems, however, as if the compiler/installation dependent time measurement routine is more ‘accurate’ for the PowerStation than for Visual Fortran 5.0.

For the large, sparse eigenproblems of Chapter 7, automatic node renumbering is applied to all problems, regardless of storage format (skyline or sparse). Again this is of interest only if ‘absolute’ times and storage requirements are considered. For comparisons and relative efficiencies of different algorithms and/or specific parts of algorithms this should make no difference since the same renumbering
NUMERICAL RESULTS

schemes are used, resulting in identical system matrices (\(K\) and \(M\)) for a given test frame, irrespective of method of solution. This assumes of course the same storage scheme. For comparison of results obtained by different storage schemes, that is skyline and sparse, the effectiveness of the renumbering, which is different for the two schemes, will also influence the results.

Finally, it should be mentioned that all subroutines assume only one level of memory. In other words, all matrices are assumed to be stored in primary (RAM) storage, or automatically brought into primary storage by the operating system's virtual memory management system - paging may (depending on the amount of available RAM) take place, but that is completely outside the control of the SAM subroutines.
4.4 Of course, the results may be different on different operating systems. However, the general trend is that the operating system with the latest security patches and updates is more secure than the one with older or missing patches. It is important to keep your operating system up to date to mitigate risks.

Software

All software and applications are tested in standard scenarios 77, and all testing protocols are standardized and documented in detail. The reports that all statistical procedures are performed with 10 independent trials of data 50 times.

For Chapters 1 through 8, all programs are built on Microsoft's Windows operating system. All programs were developed using Visual Studio 2005. The latest version of the data is available for download from the Microsoft website. These tools are all built on Microsoft's operating system and are seamlessly integrated. It is important to note that the experimental installations are dependent on the current operating system version. For the Pentium 4 system, the Visual Studio 2005 is used.

For the data, some algorithms are at Chapter 7, automated code. The results may vary depending on the specific version of the software used. It is important to keep your operating system up to date to mitigate risks.
This chapter presents results obtained by three subroutines developed to find all eigenvalues and, optionally, all eigenvectors of a full, symmetric matrix.

The matrix used is the stiffness matrix $(K)$ of the test frame described in Chapter 4. The frames have the same geometric and material properties as the test frames defined by Tables 4.1 and 4.2, however, the number of bays and stories differ, and the labeling of Chapter 4 is not used in this chapter.

The stiffness matrix is converted from a 'skyline'-stored matrix to a full, symmetric matrix. Both stocky and slender frames are considered, producing matrices with both large and small 'bandwidths'. In all cases the matrices are sparsely populated (that is, they have relatively few non-zero elements).

Both supported (fixed) and unsupported frames are considered. The unsupported frames include the rigid body displacement modes, and the eigenroutines should be capable of picking up the corresponding zero eigenvalues.

Matters related to both accuracy, robustness and efficiency are investigated, with the emphasis on efficiency.

Also included in this chapter is a section on the matrix condition number, the aim of which is to indicate the merits of a simple, approximate formula.

All results presented in this chapter are obtained by Computer III.
5.1 The subroutines

The SAM library offers three subroutines for the special, symmetric eigenproblem stated by Eq. (1.2): HQLS, HQRIS and SPSJAC. A brief description is given below; for more detailed information Chapters 1 and 2 should be consulted.

Subroutine HQLS

This subroutine computes all eigenvalues and all or none eigenvectors of a real, symmetric \( n \times n \) matrix. The full matrix is input information to the subroutine.

The solution is obtained through two steps:

1. Householder reduction to tridiagonal form (by subroutine HOUS3D).
2. Implicit QL iteration (by subroutine QLS3D or QLS3DV).

For the implicit QL iteration the EISPACK system [16] has been consulted. Subroutine QLS3D is thus based on the EISPACK routine IMTQL1, and subroutine QLS3DV is very similar to EISPACK routine IMTQL2.

If eigenvectors are computed they should, by virtue of the algorithm, satisfy the orthogonality requirement of Eq. (1.15), also in the case of multiple eigenvalues.

There is no tolerance value available for the user to specify. A maximum of 30 iterations are used.

Subroutine HQRIS

This subroutine is quite similar to HQLS. It also computes all eigenvalues and all or none eigenvectors of a real, symmetric \( n \times n \) matrix, and the full matrix is input information to the subroutine.

The main difference between the two subroutines is the way in which the eigenvectors are determined; here inverse iteration is used.

The solution is obtained in three steps:

1. Householder reduction to triangular form (by subroutine HOUS3D as for HQLS).
2. Kahan-Varah QR iteration for the eigenvalues (by subroutine QRS3D).
3. Inverse iteration for the eigenvectors (by subroutine IIS3D).
Steps 2 and 3 of this subroutine are based on a similar subroutine developed by C. A. Felippa at UC Berkeley in the latter part of the sixties.

If multiple eigenvalues are not present, orthogonal eigenvectors are computed. If multiple eigenvalues are present, the subroutine will still return distinct eigenvectors, but they will not be orthogonal.

Again there is no tolerance value available for the user to specify. QR iterations are carried out to an accuracy fairly close to (within a few digits from) machine precision.

Subroutine SPSJAC

This subroutine has the same task as the other two, that is to compute all eigenvalues and all or none eigenvectors of a real, symmetric \( nxn \) matrix. Again the full matrix is input information to the subroutine.

However, this subroutine uses quite a different algorithm. A special, cyclic Jacobi procedure reduces the matrix all the way to (a similar) diagonal form. The eigenvectors are computed from the Jacobi rotation matrices, see Eq. (2.6), and they satisfy orthogonality, Eq. (1.15), also in the case of multiple eigenvalues.

For this subroutine the user may specify an error tolerance value, \( \varepsilon \), which controls the computations as follows: At each cycle the numerically largest on- and off-diagonal matrix terms, \( A_{ij}^{\text{max}} \) and \( A_{ij}^{\text{max}} \), respectively, are determined. Computations (rotations) are aborted when

\[
err = (n - 1)\frac{|A_{ij}^{\text{max}}|}{|A_{kk}^{\text{max}}|} < \varepsilon
\]

The implication of this cut-off criteria is demonstrated in Section 5.3 below.

5.2 Accuracy and robustness

A fair number of numerical analyses have been carried out using the complete stiffness matrix of various test frames. The matrix dimension (\( n \)) varied from about 100 to almost 1100. In all cases the eigenvalues were computed with very good accuracy.

For SPSJAC (Jacobi) an error tolerance of \( 10^{-10} \) (see above) was specified in all cases.

Here we present only a few typical results, and we start with a supported non-symmetric, 'stocky' frame with \( M=N=P=4 \) resulting in a matrix dimension of \( n = 600 \).
For the given eigenvalue ordering, which is one of increasing numerical value, three typical eigenvalues, including the lowest and the highest plus one in the middle, are as follows for the three subroutines:

<table>
<thead>
<tr>
<th></th>
<th>lst eigenvalue</th>
<th>300th eigenvalue</th>
<th>600th eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>HQLS</td>
<td>1.30788329298259E+09</td>
<td>1.17171044693420D+10</td>
<td></td>
</tr>
<tr>
<td>HQRIS</td>
<td>1.30788329298259E+09</td>
<td>1.17171044693420D+10</td>
<td></td>
</tr>
<tr>
<td>SPSJAC</td>
<td>1.30788329298259E+09</td>
<td>1.17171044693417D+10</td>
<td></td>
</tr>
</tbody>
</table>

For the first two subroutines the higher eigenvalues are identical, to 15 digits, and also SPSJAC predicts the highest eigenvalue with a 13 (or perhaps 14) digits accuracy, which is better than the specified error tolerance suggests.

For the eigenvectors $Q$ the accuracy is 'measured' by some characteristics of the product $Q^T Q$. For the same problem as the above eigenvalues are obtained for, these characteristics are:

For HQLS:

SOME CHARACTERISTICS OF THE MATRIX
eigenvectors-transpose * eigenvectors:

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 600)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000027E+00</td>
<td>500 500</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999991E-01</td>
<td>520 520</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>4.454381813869976E-15</td>
<td>516 512</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>199  2</td>
</tr>
</tbody>
</table>

For HQRIS:

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 600)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000027E+00</td>
<td>409 409</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999991E-01</td>
<td>491 491</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.121978516915622E-13</td>
<td>358 357</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>88   2</td>
</tr>
</tbody>
</table>

For SPSJAC:

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 600)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>9.999999999999991E-01</td>
<td>120 120</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999991E-01</td>
<td>583 583</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>4.517219931443606E-15</td>
<td>80   47</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>80   5</td>
</tr>
</tbody>
</table>

It should be emphasized that the above results are all obtained by multiplying the eigenvector matrices, as returned by the subroutines, by their own transpose (there is no 'postprocessing'). For all
NUMERICAL RESULTS

subroutines the results seem to indicate good accuracy. It is hard to make a distinction between the subroutines, and perhaps unfair to rank them, but it appears that HQLS produces slightly better results than SPSJAC, and that they both are slightly better than HQRIS, judged on the largest off-diagonal term. An interesting observation can be made. Whilst the largest diagonal term of $Q^TQ$ for subroutines HQLS and HQRIS are slightly larger than 1 (one), in all cases observed, this extreme value is consistently (slightly) smaller than 1 for SPSJAC.

Next we consider the same frame, but this time it is unsupported. The matrix dimension is now $n = 750$, and the matrix is singular, with a defect of 6, corresponding to the six rigid body displacement modes. The 7 lowest (numerically smallest) eigenvalues for this matrix, as obtained by the three subroutines are:

<table>
<thead>
<tr>
<th>No.</th>
<th>HQLS</th>
<th>HQRIS</th>
<th>SPSJAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.08458973105036D-06</td>
<td>-1.1132192937976D-07</td>
<td>6.0679116149235D-08</td>
</tr>
<tr>
<td>2</td>
<td>2.39885689954775D-06</td>
<td>-2.2672137898682D-07</td>
<td>8.5541800568757D-08</td>
</tr>
<tr>
<td>3</td>
<td>8.5022400721317D-06</td>
<td>8.8198547330919D-07</td>
<td>1.0586195875183D-07</td>
</tr>
<tr>
<td>4</td>
<td>1.5256916331643D-05</td>
<td>1.4750952434780D-07</td>
<td>2.2559265116846D-07</td>
</tr>
<tr>
<td>5</td>
<td>2.3641638147563D-05</td>
<td>1.715100151278E-06</td>
<td>-3.7885116508154D-07</td>
</tr>
<tr>
<td>6</td>
<td>-5.8630431662379D-05</td>
<td>5.996065068482246D-06</td>
<td>-4.5768530118987D-07</td>
</tr>
<tr>
<td>7</td>
<td>3.1209364245532D+06</td>
<td>3.1209364245510D+06</td>
<td>3.1209364245508D+06</td>
</tr>
</tbody>
</table>

For all practical purposes the first six eigenvalues are 'numerically' zero, for all three subroutines, and the first non-zero eigenvalue (number 7 in the sequence) is determined with the same accuracy as for the supported frame.

For the product $Q^TQ$, HQLS and SPSJAC give similar results as for the supported frame, that is a matrix very close to the identity matrix. HQRIS, however, now fails to produce eigenvectors that satisfy the orthogonality condition.

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 750)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element: 1.000000000000008E+00</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Smallest diagonal element: 9.999999999999952E-01</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Largest off-diag. element: 9.661433564894345E-01</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Smallest off-diag. element: 0.000000000000000E+00</td>
<td>481</td>
<td>100</td>
</tr>
</tbody>
</table>

This was to be expected since the matrix now contains multiple eigenvalues (six eigenvalues are numerically equal to zero).

The last problem we consider is that of a symmetric, slender and supported frame. The frame has one bay in each direction and 20 stories, that is $M = N = 1$ and $P = 20$. Supported (fixed) at the bottom, this gives a matrix dimension of $n = 480$.  

Accuracy and robustness 75
This problem has, due to the symmetry, many multiple eigenvalues.
The first 11 (numerically smallest) eigenvalues obtained by the three subroutines are as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>HQLS</th>
<th>HQRIS</th>
<th>SPSJAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.50385745354306 D+04</td>
<td>4.50385745359029 D+04</td>
<td>4.50385745350635 D+04</td>
</tr>
<tr>
<td>2</td>
<td>4.50385745416021 D+04</td>
<td>4.5038574537048 D+04</td>
<td>4.50385745358922 D+04</td>
</tr>
<tr>
<td>3</td>
<td>2.94548175106940 D+04</td>
<td>2.94548175106897 D+05</td>
<td>2.94548175105411 D+05</td>
</tr>
<tr>
<td>4</td>
<td>8.82260242509987 D+05</td>
<td>8.82260242513971 D+05</td>
<td>8.82260242514997 D+05</td>
</tr>
<tr>
<td>5</td>
<td>8.82260242514997 D+05</td>
<td>8.82260242515000 D+05</td>
<td>8.82260242515017 D+05</td>
</tr>
<tr>
<td>6</td>
<td>2.65717004882292 D+06</td>
<td>2.65717004882207 D+06</td>
<td>2.65717004882215 D+06</td>
</tr>
<tr>
<td>7</td>
<td>4.04847080401199 D+06</td>
<td>4.04847080401716 D+06</td>
<td>4.04847080401760 D+06</td>
</tr>
<tr>
<td>8</td>
<td>4.04847080401689 D+06</td>
<td>4.04847080401922 D+06</td>
<td>4.04847080401764 D+06</td>
</tr>
<tr>
<td>9</td>
<td>7.41512163269540 D+06</td>
<td>7.41512163269600 D+06</td>
<td>7.41512163269605 D+06</td>
</tr>
<tr>
<td>10</td>
<td>9.53862079503095 D+06</td>
<td>9.53862079504102 D+06</td>
<td>9.53862079504249 D+06</td>
</tr>
<tr>
<td>11</td>
<td>9.53862079504154 D+06</td>
<td>9.53862079504215 D+06</td>
<td>9.53862079504249 D+06</td>
</tr>
</tbody>
</table>

None of the subroutines have any problems in picking up the multiple eigenvalues, but it appears that the Jacobi algorithm does a slightly better job than the other two on this problem. However, from a practical point of view there is little in it.

The eigenvector product $Q^TQ$ is, for all practical purposes, equal to the identity matrix for both HQLS and SPSJAC - the accuracy is roughly the same as for the 'stocky' frame (with $n = 600$). For HQRIS on the other hand, the orthogonality condition is not met, as can be seen from the following characteristics of $Q^TQ$:

<table>
<thead>
<tr>
<th>EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 480)</th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000000E+00</td>
<td>380</td>
<td>380</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999999E-01</td>
<td>357</td>
<td>357</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>9.999999999999999E-01</td>
<td>154</td>
<td>153</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>69</td>
<td>9</td>
</tr>
</tbody>
</table>

5.3 Computational efficiency

In order to establish some guidelines on the relative efficiency of the algorithms used, a series of computations have been carried out. The matrices analysed are all stiffness matrices of the test frame described in Chapter 4. $M$ and $N$ designate the number of bays in the $x$- and $y$-direction, respectively, whereas $P$ designates the number of stories. The corresponding matrix dimension is designated by $n$.

All results presented in this section are obtained with Computer III, see Section 4.4, and we give the actual measured CPU-times in sec-
NUMERICAL RESULTS

Most problems are analyzed twice, once for the eigenvalues only, and once for both eigenvalues and eigenvectors.

Table 5.1 gives measured CPU-times for finding all eigenvalues and corresponding eigenvectors of the stiffness matrices of supported, 'stocky', non-symmetric frames.

<table>
<thead>
<tr>
<th>Frame</th>
<th>CPU-time [seconds] - Computer III</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HQLS</td>
</tr>
<tr>
<td>2/2/2</td>
<td>108</td>
</tr>
<tr>
<td>3/3/3</td>
<td>288</td>
</tr>
<tr>
<td>4/4/4</td>
<td>600</td>
</tr>
<tr>
<td>5/5/5</td>
<td>1080</td>
</tr>
</tbody>
</table>

Corresponding eigenvectors of the stiffness matrices of 4 supported and non-symmetrical frames of the 'stocky' type. Since the frames are supported the stiffness matrices are regular and positive definite, leaving all eigenvalues greater than zero. For each frame and subroutine two numbers are given, separated by a slash (/). The first number is the time required to find only the eigenvalues, whereas the second number is the time required for finding both the eigenvalues and the corresponding eigenvectors.

Table 5.2 gives the results for the first three frames of Table 5.1, but this time the frames are unsupported. Hence, the corresponding stiffness matrices include more degrees of freedom (n is greater). Since the rigid body displacements are also included the matrices are singular and semi-definite, each having 6 zero eigenvalues, one for each rigid body mode.

<table>
<thead>
<tr>
<th>Frame</th>
<th>CPU-time [seconds] - Computer III</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HQLS</td>
</tr>
<tr>
<td>2/2/2</td>
<td>162</td>
</tr>
<tr>
<td>3/3/3</td>
<td>384</td>
</tr>
<tr>
<td>4/4/4</td>
<td>750</td>
</tr>
</tbody>
</table>

Computational efficiency
PART II

From Tables 5.1 and 5.2 the following observations can be made:

- For finding eigenvalues only, the QL and QR algorithms used by HQLS and HQRIS, respectively, are almost equally efficient (HQRIS is consistently faster, but only just), and they are both far superior to the Jacobi algorithm of SPSJAC.

- When eigenvectors are also computed the measurements are clearly in favor of HQRIS, that is in favor of inverse (power) iterations over implicit QL iterations, roughly by a factor of two. Again Jacobi is a much 'slower' process - there is almost an order of magnitude between HQRIS and SPSJAC.

- The time measured by SPSJAC (Jacobi) for finding the eigenvalues of the matrix corresponding to the largest frame of Table 5.1 (n = 1080) is difficult to explain. Judging from the other times measured one would have expected a time in the neighborhood of 3000 seconds. Instead 10500 second was measured, and this is the lower one of two times measured for the same computation. It is tempting to put this down to the computer configuration (cache, paging etc.) - the Jacobi algorithm accesses the matrix elements far more randomly than does the Householder tridiagonalization procedure used by the other two subroutines - but there is no hard evidence to support this suggestion. The question mark in Table 5.1 simply indicates that this computation was not attempted.

Computations were also carried out for stiffness matrices of supported frames of the slender type which produce more diagonal dominant matrices with higher condition numbers. Table 5.3 shows results obtained for non-symmetric frames.

**Table 5.3:** CPU-times for finding eigenvalues / eigenvalues + eigenvectors of stiffness matrices of supported, slender, non-symmetric frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>M/N/P</th>
<th>n</th>
<th>CPU-time [seconds] - Computer III</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>HQLS</td>
</tr>
<tr>
<td>1/1/5</td>
<td></td>
<td>120</td>
<td>0.281 / 1.43</td>
</tr>
<tr>
<td>1/1/10</td>
<td></td>
<td>240</td>
<td>2.37 / 11.3</td>
</tr>
<tr>
<td>1/1/20</td>
<td></td>
<td>480</td>
<td>26.4 / 106</td>
</tr>
<tr>
<td>1/1/40</td>
<td></td>
<td>960</td>
<td>236 / 894</td>
</tr>
</tbody>
</table>

Again we see an unexpectedly large number (well over two hours) for the Jacobi algorithm when applied to the tallest frame (n = 960).

Some of the computations of Table 5.3 were repeated for a symmetrical, slender and supported frame. Results are shown in Table 5.4.
NUMERICAL RESULTS

Table 5.4: CPU-times for finding eigenvalues and eigenvectors of stiffness matrices of supported, slender and symmetric frames

<table>
<thead>
<tr>
<th>Frame M/N/P</th>
<th>n</th>
<th>CPU-time [seconds] - Computer III</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/10</td>
<td>240</td>
<td>HQLS: 10.1; HQRIS: 4.67; SPSJAC: 35.0</td>
</tr>
<tr>
<td>1/1/20</td>
<td>480</td>
<td>HQLS: 99.5; HQRIS: 46.7; SPSJAC: 350</td>
</tr>
<tr>
<td>1/1/40</td>
<td>960</td>
<td>HQLS: 851; HQRIS: 426; SPSJAC: 12900</td>
</tr>
</tbody>
</table>

It should be noted that times are only given for finding both eigenvalues and eigenvectors. Hence the numbers of Table 5.4 should be compared with the higher of the two numbers quoted for each frame and subroutine in Table 5.3. This comparison reveals no major surprises. There seems to be no extra costs involved for finding multiple eigenvalues; on the contrary, for HQLS, and particularly for HQRIS, times for the largest frame are in fact appreciably lower. Again we notice a very large time for the Jacobi algorithm in case of the tallest frame.

The last computations we report on concerns the error tolerance value $\varepsilon$ specified for subroutine SPSJAC (see Section 5.1). So far all analyses have been carried out with $\varepsilon = 10^{-10}$. Five computations were carried out for the slender, symmetrical and supported frame with one bay in each direction ($M=N=1$) and ($P=20$) stories, that is for a regular matrix with $n = 480$. For values of $\varepsilon$ equal to $10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}$ and $10^{-12}$, the following CPU-times, in seconds, were recorded for finding both eigenvalues and eigenvectors: 216, 271, 319, 337 and 372, respectively. In order to assess the meaning of the 'error test' described in Section 5.1, we present three eigenvalues, the lowest (number 1), a medium (number 200) and the highest (number 480), computed for different values of $\varepsilon = \text{EPS}$:

<table>
<thead>
<tr>
<th>EPS</th>
<th>1st eigenvalue</th>
<th>200th eigenvalue</th>
<th>480th eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0D-04</td>
<td>4.50405150802971D+04</td>
<td>9.25854470014383D+08</td>
<td>1.28088547246036D+10</td>
</tr>
<tr>
<td>10.0D-05</td>
<td>4.50385745622424D+04</td>
<td>9.25854469478485D+08</td>
<td>1.2808854724609D+10</td>
</tr>
<tr>
<td>10.0D-08</td>
<td>4.50385745350681D+04</td>
<td>9.25854469478499D+08</td>
<td>1.2808854724609D+10</td>
</tr>
<tr>
<td>10.0D-10</td>
<td>4.50385745350635D+04</td>
<td>9.25854469478499D+08</td>
<td>1.2808854724609D+10</td>
</tr>
<tr>
<td>10.0D-12</td>
<td>4.50385745350635D+04</td>
<td>9.25854469478499D+08</td>
<td>1.2808854724609D+10</td>
</tr>
</tbody>
</table>

These results are somewhat surprising, and it appears that we have perhaps been a bit unfair to the Jacobi algorithm, using an unnecessarily low value for $\varepsilon = 10^{-10}$ in our comparisons with the other two algorithms. Even for the highest value, $\varepsilon = 10^{-4}$, the largest eigenvalue seems to be accurate to about 12 digits, with the lowest
being accurate to about 4 digits. For $\varepsilon = 10^{-6}$, the lowest eigenvalue is accurate to about 9 digits, and the largest eigenvalue is now accurate to all 15 digits. From this it appears that an $\varepsilon$-value in the neighborhood of $10^{-6}$ should be adequate for most purposes. This statement is supported by the characteristics of the eigenvector product $Q^TQ$.

For $\varepsilon = 10^{-4}$ we find:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>9.999999999999983E-01</td>
<td>166</td>
<td>166</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.9999999999999813E-01</td>
<td>469</td>
<td>469</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>1.805933112336002E-15</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>272</td>
<td>3</td>
</tr>
</tbody>
</table>

For $\varepsilon = 10^{-12}$ we find:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>9.999999999999908E-01</td>
<td>191</td>
<td>191</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999648E-01</td>
<td>314</td>
<td>314</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>2.5604518515041767E-15</td>
<td>134</td>
<td>130</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>243</td>
<td>25</td>
</tr>
</tbody>
</table>

For practical purposes there is no difference in these results (if anything the eigenvector product for the largest $\varepsilon$-value is the one closest to the identity matrix!).

The savings in computational effort, by increasing $\varepsilon$ from $10^{-10}$ to $10^{-6}$, are significant.

### 5.4 The matrix condition number

The matrix condition number, defined as

$$\kappa = |\lambda_{\text{max}}/\lambda_{\text{min}}|$$

is a useful property for assessing numerical problems that may have been encountered when subjecting the matrix to numerical operations, such as factorization. However, the exact determination of $\kappa$, requiring both $\lambda_{\text{max}}$ and $\lambda_{\text{min}}$ to be computed, is very costly and it is seldom, if ever, undertaken unless it comes as a by-product of other computations. For practical purposes one does not require $\kappa$ to be known with great accuracy, the order of magnitude is usually sufficient. Hence the interest in approximations.
NUMERICAL RESULTS

The trace of a matrix $A$, $\text{tr}(A)$, which is readily available, see Eq. (1.13), is usually considered a good estimate of $\lambda_{\text{max}}$. Unfortunately, we do not have an equally simple and good estimate for $\lambda_{\text{min}}$. A sometimes quoted estimate for $\lambda_{\text{min}}$ is the (numerically) smallest element, $|D_{i\text{min}}|$, of the diagonal matrix $D_A$ in the factorization $A = L_A D_A L_A^T$. This gives us the following approximate expression for the condition number:

$$\kappa_{\text{approx}} = \frac{\text{tr}(A)}{|D_{i\text{min}}|}$$

In order to evaluate the effectiveness of this approximation, a series of computations, which determined both the exact and the approximate condition numbers for the stiffness matrix of various test frames, were carried out. The results are shown in Table 5.5.

**Table 5.5:** Exact and approximate matrix condition numbers evaluated for the stiffness matrix of various test frames

<table>
<thead>
<tr>
<th>Frame $M - N - P$</th>
<th>$n$</th>
<th>Matrix condition numbers</th>
<th>$\kappa_{\text{exact}}$</th>
<th>$\kappa_{\text{approx}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 - 2 - 2</td>
<td>108</td>
<td>$5,02 \times 10^2$</td>
<td>$2,13 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>3 - 3 - 3</td>
<td>288</td>
<td>$1,19 \times 10^3$</td>
<td>$6,88 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>4 - 4 - 4</td>
<td>600</td>
<td>$2,16 \times 10^3$</td>
<td>$1,50 \times 10^4$</td>
<td></td>
</tr>
<tr>
<td>5 - 5 - 5</td>
<td>1080</td>
<td>$3,39 \times 10^3$</td>
<td>$2,71 \times 10^4$</td>
<td></td>
</tr>
<tr>
<td>1 - 1 - 5</td>
<td>120</td>
<td>$4,68 \times 10^3$</td>
<td>$1,24 \times 10^4$</td>
<td></td>
</tr>
<tr>
<td>1 - 1 - 10</td>
<td>240</td>
<td>$2,83 \times 10^4$</td>
<td>$7,96 \times 10^4$</td>
<td></td>
</tr>
<tr>
<td>1 - 1 - 20</td>
<td>480</td>
<td>$2,80 \times 10^5$</td>
<td>$7,67 \times 10^5$</td>
<td></td>
</tr>
<tr>
<td>1 - 1 - 40</td>
<td>960</td>
<td>$3,72 \times 10^6$</td>
<td>$9,40 \times 10^6$</td>
<td></td>
</tr>
</tbody>
</table>

We see that the approximate formula consistently overestimates the correct value (which is basically good news), and it gives, by and large, the correct order of magnitude. The number of comparisons is not large enough to make a sweeping statement, but it appears that $\kappa_{\text{approx}}$ provides a good indication of the condition number of a matrix.
5.5 Concluding remarks

All three subroutines available for the special eigenproblem seem to behave quite well. In terms of robustness and accuracy it is hard to distinguish between them. If eigenvectors are required, HQLS and SPSJAC may seem to have the edge, in that they both will return orthogonal eigenvectors, also in the case of multiple eigenvalues.

In terms of computational efficiency the two QL/QR based subroutines, HQLS and HQRIS, are almost equal if only eigenvalues are requested (with HQRIS having a slight advantage), and they are far superior to the Jacobi based routine, SPSJAC. However, if both eigenvalues and eigenvectors are requested, HQRIS, which uses inverse iteration to find the eigenvectors, is the clear winner. But, as pointed out, it fails to return orthogonal eigenvectors in the case of multiple eigenvalues. This may not be a serious problem since it always returns distinct eigenvectors which may be made orthogonal by some postprocessing operation like modified Gram-Schmidt. It is quite possible, in fact very likely (but not investigated), that HQRIS will come out as the most efficient subroutine even with such a postprocessing of the eigenvectors included.

It should be pointed out that SPSJAC (Jacobi) is not as inefficient as the comparisons may seem to indicate. It can be speeded up, without much loss of accuracy, by increasing the tolerance value used in all comparison analyses. Nevertheless, it still remains much slower than the other two subroutines.

A simple and frequently used approximation for the condition number of a matrix is shown to provide quite a reasonable estimate of this important numerical indicator.
This chapter presents results obtained by two subroutines developed to find all eigenvalues and, optionally, all eigenvectors of the general eigenproblem defined by Eq. (1.1), for full matrices A and B.

The test problems considered are free vibration problems of the test frame described in Chapter 4. Matrices A and B are the stiffness and mass matrices of the frame, converted from 'skyline' storage format to full matrices. The frames have the same geometric and material properties as the test frames defined by Tables 4.1 and 4.2; however, as in Chapter 5, the number of bays and stories differ, and the labeling of Chapter 4 is not used in this chapter.

Both stocky and slender frames are considered, producing matrices with both large and small 'bandwidths'. Symmetric frames, having multiple eigenvalues, are also included in the test program. In all cases the matrices are sparsely populated (that is, they have relatively few non-zero elements).

Matters related to both accuracy, robustness and efficiency are investigated, with the emphasis on efficiency.

All results presented in this chapter are obtained by Computer III.
6.1 The subroutines

The SAM library includes two subroutines for the solution of small, general, symmetric eigenproblems involving full matrices: GENHQL and GENJAC.

A brief description is given below; for more detailed information Chapters 1 and 2 should be consulted.

Subroutine GENHQL

This subroutine computes all eigenvalues, \( \lambda_i \), and all or none of the corresponding eigenvectors of the generalized, symmetric eigenproblem

\[
(A - \lambda B)q_i = 0
\]

where \( A \) and \( B \) are full, symmetric matrices, and \( B \) is also positive definite.

The solution proceeds in four steps:

1. The problem is transformed to special form through Cholesky factorization of \( B \), see Section 1.11 (by subroutines FACHOL and FSCHOL).
2. The special problem is reduced to tridiagonal form by Householder transformation, see Section 2.2 (by subroutine HOUS3D).
3. Eigenvalues and (if requested) eigenvectors are extracted by implicit QL iteration, see Section 2.2 (by subroutines QLS3D or QLS3DV).
4. Eigenvectors are transformed back to the original problem, if relevant, see Section 1.11 (by subroutine BSCHOL).

If eigenvectors are computed they should, by virtue of the algorithm, satisfy the orthogonality requirement \( Q^\top BQ = I \), also in the case of multiple eigenvalues.

There is no tolerance value available for the user to specify. A maximum of 30 QL iterations are used.

Subroutine GENJAC

This subroutine has the same task as GENHQL, that is to compute all eigenvalues and all or none eigenvectors of the generalized, symmetric eigenproblem of dimension \( n \).

A generalized Jacobi procedure, due to FALK and LANGEMEYER [6], is used, in which the simultaneous transformations of \( A \) and \( B \)
towards diagonal form proceed in sweeps. Each sweep consists of a series of elementary, non-orthogonal transformations of the form

\[ A^{(k+1)} = T_k A^{(k)} T_k \quad \text{and} \quad B^{(k+1)} = T_k B^{(k)} T_k \]

The transformation matrix \( T_k \) contains only unit elements on the diagonal, and the only non-zero off-diagonal elements, \( T_{ij} \) and \( T_{ji} \), are selected in such a way as to reduce to zero simultaneously elements \((i,j)\) and \((j,i)\) of \( A^{(k)} \) and \( B^{(k)} \).

At the beginning of each sweep \( A^{(k)} \) and \( B^{(k)} \) are normalized by pre- and post-multiplication with a diagonal matrix whose elements are the inverse square roots of \( B_{ii}^{(k)} \) such that after normalization the diagonal elements of \( B^{(k)} \) are equal to unity.

For the normalized matrices the quantities

\[ \varepsilon_A = n \max \left\{ \frac{|A_{ij}^{(k+1)}|}{\sum |A_{ii}^{(k+1)}|} \right\} \quad \text{and} \quad \varepsilon_B = \max \left\{ |B_{ji}^{(k+1)}| \right\} \quad i \neq j \]

are computed. The convergence criteria is that

\[ \max \{ \varepsilon_A, \varepsilon_B \} < \varepsilon \]

where \( \varepsilon \) is a specified tolerance. If this requirement is not satisfied, new bounds (or thresholds) are determined for the off-diagonal terms of \( A^{(k+1)} \) and \( B^{(k+1)} \), and during the next sweep, transformations are carried out until all off-diagonal elements are below the thresholds.

### 6.2 Accuracy and robustness

All test problems analysed and reported in this section use a consistent mass matrix \( (B) \).

We first investigate the tolerance parameter, \( \varepsilon \), of GENJAC. The problem is a 'stocky', non-symmetric frame with \( M=N=P=2 \), resulting in \( n = 108 \). Eigenvalues and eigenvectors are computed by both GENHQL and GENJAC. For the latter, computations were carried out for five values of \( \varepsilon \), \( 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12} \) and \( 10^{-14} \).

Three typical eigenvalues, including the lowest, \( \lambda_1 \), and the highest, \( \lambda_{108} \), plus one in the middle, \( \lambda_{50} \), are as follows, for the different \( \varepsilon \)-values:
The last line includes the results obtained by GENHQL.

Apart from the two lowest values of ε (=EPS), the results are apparently very similar to those obtained by SPSJAC in the previous chapter. However, if we also look at the product $Q^T B Q$, where $Q$ is the eigenvector matrix returned by the subroutines (if requested), the picture is slightly different. The numerically extreme values of this product, as obtained with GENJAC for the different values of $\epsilon$, are as follows:

For $\epsilon = 10^{-4}$:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000003E+00</td>
<td>66</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999965E-01</td>
<td>79</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.528459275036193E-05</td>
<td>100</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>4.857225732735060E-17</td>
<td>96</td>
</tr>
</tbody>
</table>

For $\epsilon = 10^{-6}$:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000004E+00</td>
<td>30</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999969E-01</td>
<td>26</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.514367964239071E-07</td>
<td>54</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>4.141522988946277E-17</td>
<td>67</td>
</tr>
</tbody>
</table>

For $\epsilon = 10^{-8}$:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000004E+00</td>
<td>62</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999962E-01</td>
<td>52</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.509470678970787E-09</td>
<td>90</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>3.469446951953614E-18</td>
<td>90</td>
</tr>
</tbody>
</table>

For $\epsilon = 10^{-10}$:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000004E+00</td>
<td>62</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999961E-01</td>
<td>52</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.481425028099675E-11</td>
<td>66</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>65</td>
</tr>
</tbody>
</table>

For $\epsilon = 10^{-12}$:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000004E+00</td>
<td>62</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999964E-01</td>
<td>52</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.42725402253213E-13</td>
<td>35</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>70</td>
</tr>
</tbody>
</table>
NUMERICAL RESULTS

For $\epsilon = 10^{-14}$:

<table>
<thead>
<tr>
<th>Value</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000004E+00</td>
<td>62</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999961E-01</td>
<td>52</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.494005416219807E-15</td>
<td>74</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>81</td>
</tr>
</tbody>
</table>

We also include the same values obtained by GENHQL:

<table>
<thead>
<tr>
<th>Value</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000007E+00</td>
<td>108</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999976E-01</td>
<td>8</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>1.941589250487041E-15</td>
<td>108</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>92</td>
</tr>
</tbody>
</table>

We see that the largest off-diagonal element of the matrix $Q^T B Q$, as obtained by GENJAC, is of the same order of magnitude as the tolerance value $\epsilon$, and apparently GENJAC computes the eigenvalues with higher accuracy than the eigenvectors. We also notice that in order for GENJAC to obtain eigenvectors with the same accuracy as GENHQL, we need to specify a value of $10^{-14}$ for $\epsilon$.

All subsequent results obtained by GENJAC, and presented in this section, correspond to a tolerance value of $\epsilon = 10^{-10}$.

We next consider a symmetric problem with multiple eigenvalues. It is again a ‘stocky’ frame, but slightly larger, with $M=N=P=3$ resulting in $n = 288$. Some typical eigenvalues computed are as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>GENHQL</th>
<th>GENJAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.98711998692331D+03</td>
<td>1.98711998692253D+03</td>
</tr>
<tr>
<td>2</td>
<td>1.98711998692396D+03</td>
<td>1.98711998692272D+03</td>
</tr>
<tr>
<td>3</td>
<td>2.54742192740366D+03</td>
<td>2.54742192740227D+03</td>
</tr>
<tr>
<td>4</td>
<td>1.3174554129085D+04</td>
<td>1.3174554129075D+04</td>
</tr>
<tr>
<td>5</td>
<td>1.98466542066979D+04</td>
<td>1.98466542066966D+04</td>
</tr>
<tr>
<td>6</td>
<td>1.98466542067008D+04</td>
<td>1.98466542066966D+04</td>
</tr>
<tr>
<td>100</td>
<td>7.83619597739667D+05</td>
<td>7.83619597739666D+05</td>
</tr>
<tr>
<td>288</td>
<td>8.45924121609989D+06</td>
<td>8.45924121609993D+06</td>
</tr>
</tbody>
</table>

No surprises here. Both subroutines pick up multiple eigenvalues without any problems. The eigenvector product $Q^T B Q$ also shows that $B$-orthogonal eigenvectors are determined with the same level of accuracy as for the non-symmetric frame:

GENHQL:

<table>
<thead>
<tr>
<th>Value</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000016E+00</td>
<td>55</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999938E-01</td>
<td>97</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>2.379931087857204E-15</td>
<td>183</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>258</td>
</tr>
</tbody>
</table>
The last problem we consider is a 'slender', non-symmetric frame with \( M = N = 1 \) and with varying \( P \) (number of stories). For \( P = 10 \) all results are normal. However, for \( P = 20 \) we have a major problem with GENJAC, it breaks down (due to a very small element encountered on the diagonal of the transformed \( B \)-matrix). This breakdown is independent of the tolerance value \( \varepsilon \).

It turns out that the largest value of \( P \) that GENJAC produces results for is 18 (corresponding to \( n = 432 \)). For this value everything seems to be normal, eigenvalues and eigenvectors are computed with expected accuracy for several values of \( \varepsilon \), but for \( P = 19 \) the algorithm breaks down. This seems to indicate that the ‘breakdown’ test may be questionable.

GENHQL has no problem with \( P = 20 \), in fact it produces ‘normal’ results also for \( P = 40 \) (\( n = 960 \)), in which case the characteristics of matrix \( Q^TBQ \), for instance, is:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000000000047E+00</td>
<td>957</td>
<td>957</td>
</tr>
<tr>
<td>9.999999999999902E-01</td>
<td>940</td>
<td>940</td>
</tr>
<tr>
<td>7.153132253190364E-15</td>
<td>849</td>
<td>848</td>
</tr>
<tr>
<td>0.000000000000000E+00</td>
<td>176</td>
<td>1</td>
</tr>
</tbody>
</table>

6.3 Computational efficiency

In order to establish the relative efficiency of the two subroutines a series of computations were carried out for test frames of the types described in Chapter 4. \( M \) and \( N \) designate the number of bays in the \( x \)- and \( y \)-direction, respectively, whereas \( P \) designates the number of stories. The corresponding matrix dimension is designated by \( n \).

All results presented in this section are obtained with Computer III, see Section 4.4, for consistent mass matrices, and unless specifically stated otherwise the tolerance parameter of GENJAC, \( \varepsilon \), is always \( 10^{-10} \). Also most problems are analysed twice, once for the eigenvalues only, and once for both eigenvalues and eigenvectors.

Table 6.1 gives measured CPU-times for finding eigenvalues and eigenvectors, for six test frames, including both ‘stocky’ and ‘slender’ frames. For each frame and subroutine two numbers are given,
Table 6.1: CPU-times for finding eigenvalues / eigenvalues +
eigenvectors of some non-symmetric test frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>$M/N/P$</th>
<th>$n$</th>
<th>CPU-time [seconds] - Computer III</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN/PI</td>
<td>GENHQL</td>
<td>GENJAC</td>
<td></td>
</tr>
<tr>
<td>2 / 2 / 2</td>
<td>108</td>
<td>0.379 / 1.26</td>
<td>5.93 / 8.40</td>
</tr>
<tr>
<td>3 / 3 / 3</td>
<td>288</td>
<td>9.83 / 32.2</td>
<td>222 / 253</td>
</tr>
<tr>
<td>4 / 4 / 4</td>
<td>600</td>
<td>111 / 345</td>
<td>2770 / 2940</td>
</tr>
<tr>
<td>1 / 1 / 10</td>
<td>240</td>
<td>4.94 / 16.4</td>
<td>110 / 140</td>
</tr>
<tr>
<td>1 / 1 / 18</td>
<td>432</td>
<td>37.8 / 125</td>
<td>840 / 971</td>
</tr>
<tr>
<td>1 / 1 / 40</td>
<td>960</td>
<td>496 / 1570</td>
<td>Breakdown</td>
</tr>
</tbody>
</table>

separated by a slash (/). The first number is the time required to find
only the eigenvalues, whereas the second number is the time
required to find both eigenvalues and corresponding eigenvectors.

Table 6.1 shows that the generalized Jacobi algorithm, as imple­
mented in GENJAC, is far less efficient than its opponent for the
job, GENHQL. In terms of eigenvalues only there is, except for the
smallest problem, a factor of more than 20 in favor of GENHQL.
This factor falls to about 8 when eigenvectors are also requested. In
other words, eigenvectors are, relatively speaking, far more expen­
sive for GENHQL to find than for GENJAC. Whereas GENJAC
requires only about 10 to 20 per cent more time to also find the
eigenvectors, GENHQL requires about 300 per cent more time. But
in absolute terms GENHQL still beats GENJAC, hands down.

Some of the computations in Table 6.1 were repeated for symmetric
frames with no significant differences in computing times.

The effect of the tolerance parameter, $\varepsilon$, on the computing times for
GENJAC is illustrated by the following CPU-times (in seconds),
recorded for finding both eigenvalues and eigenvectors of the small­
est problem of Table 6.1, namely $M=N=P=2$,

| $\varepsilon$ | 6.48 | 7.03 | 7.69 | 8.40 | 8.46 | 8.73 |

for $\varepsilon = 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}$ and $10^{-14}$, respectively.
These times seem to support the well-known convergence property
of the Jacobi algorithm that once a reasonable accuracy has been
attained further improvements are ‘cheap’.

89
6.4 Concluding remarks

The main purpose of this chapter has been to find the best algorithm for solving the reduced, but "full", eigenproblem encountered in the solution of large, sparse eigenproblem by the subspace iteration method. As general, 'stand alone' eigensolvers these routines are of no great importance.

Based on the results of the two previous sections the answer is quite obvious. There is, however, a complicating factor. In the subspace iteration algorithm, the reduced but 'full' subspace matrices, $A_{k+1}$ and $B_{k+1}$, of the eigenproblem in Eq. (3.8), tend to become more and more diagonal dominant as $k$ increases. This favors the Jacobi algorithm, and explains its popularity in subspace iteration implementations. Some limited testing, using both GENJAC and GENHQL in an implementation of subspace iteration, clearly indicates a much smaller difference in efficiency between the two subroutines than was demonstrated for individual ('one off') problems in the previous section. However, GENHQL was still faster (it should be kept in mind that some of the matrices for the test frames in Section 6.3 are not only sparse, but also diagonally dominant, thus favoring Jacobi).

The current implementation of the generalized Jacobi (which dates back to the early seventies) also looses on robustness. This may, however, be caused by improper or too stringent 'stability' tests. It is also quite possible that other implementations, for instance the one suggested by Vissing [10], may prove both more robust and possibly also more efficient. This should be looked into, but for the time being we settle for the GENHQL subroutine.

It can be argued that GENHQL can also be made more efficient by replacing the implicit QL algorithm with QR and inverse iteration, compare HQRIS and HQLS of Chapter 5. However, the gain in efficiency is not considered to be sufficient to compensate for a possible lack of robustness. Remember that HQRIS does not return orthogonal eigenvectors if multiple eigenvalues are present; this would require post processing by, for instance, Gram-Schmidt orthogonalization. Since the orthogonality property of the eigenvector matrix $(Q^T B Q = I)$ is vital for the use in subspace iteration, we stick with GENHQL.
This chapter presents results obtained by the two main eigensolvers of the SAM library, both of which are aimed at large and sparse symmetric matrices, $A$ and $B$, of the generalized eigenproblem. One solver is based on the subspace iteration algorithm, the other on the truncated Lanczos method. Both solvers come in two versions, depending on the method of storage. One version is for so-called ‘skyline’ (or profile) storage format and one is for a true sparse storage format. Most results presented pertain to the skyline versions; in the penultimate section some comparisons between the two versions are given.

All test problems considered deal with free vibration of the test frames described in Chapter 4 and classified in Tables 4.1 and 4.2. For each solver, matters related to accuracy, robustness, convergence and efficiency (in terms of both time and storage) are dealt with, and one section is devoted to comparison between the two solvers. The chapter is concluded by some general remarks on the practical use of the subroutines.

All four computers specified in Section 4.4 have been used for the computations. Wherever appropriate, the actual computer used will be specified.

### 7.1 The subroutines

The subroutines based on subspace iteration are MSSIT and SPRSIT for skyline and sparse storage format, respectively.
LANCZ2 and SPRLAN are the skyline and sparse version, respectively, based on the truncated Lanczos method. The 2 in LANCZ2 indicates that there is also an older version, called LANCZ1. This subroutine, along with an older version of the subspace based subroutine, SSSIT, are still maintained in the SAM library (for the sake of existing programs). However, the newer versions are believed to be better in all respects, and they are recommended for new developments.

The skyline and sparse versions are, apart from the storage format, identical (with respect to capabilities and algorithms). The arguments are the same except for the integer tables (one-dimensional arrays) defining the storage formats. Hence in the remainder of this section we only distinguish between the method of solution, that is between subspace iteration and truncated Lanczos.

Both subroutines are designed to determine the \( m \) eigenvalues \( \lambda_i \) and corresponding eigenvectors \( q_i \) of the generalized, symmetric eigenproblem of Eq. (3.1), that is

\[
AQ = BQL
\]

or

\[
(A - \lambda_i B)q_i = 0
\]

that are closest to a specified shift value \( \sigma \).

The \( n \) by \( n \) \( A \) matrix is a symmetric skyline (sparse) matrix. If the shift value \( \sigma \) is zero, the subroutines also accept the factors \( L_A/D_A \) (of the factorization \( A = L_A D_A L_A^T \)) as input (for the Lanczos method also the Cholesky factor, \( U_A \) or \( U_A \), depending on how the control parameters are set). The \( B \) matrix is a symmetric skyline (sparse) matrix, with the same skyline as \( A \), or it is a diagonal matrix.

At least one of the matrices, \( A \) or \( B \), should be positive definite, or it should be possible to make \( A \) positive definite through a suitable choice of the shift value \( \sigma \).

Both subroutines have a ‘print switch’ (an integer parameter) as input argument. This parameter controls the type and amount of
The subroutines

Numerical Results

'internal information' printed by the subroutine on a specified logical unit number. The subroutines also perform a fair number of consistency checks on input parameters, and a number of error traps are implemented. An error situation causes a message to be printed, computations to be aborted and control to be returned to the calling program, accompanied by a negative error flag (an integer variable in the argument list).

Subroutine MSSIT (SPRSIT)

This subroutine computes the eigenvalues and eigenvectors by a slightly modified subspace iteration technique (compared to the 'old' SSSIT routine). The computational procedure is described in detail in Section 3.1. The reduced, subspace eigenproblem, which involves full matrices, is solved by subroutine GENHQL (described in Chapter 6). MSSIT (SPRSIT) always returns approximations to the m eigenvectors $Q = [q_1, q_2, \ldots, q_m]$ corresponding to the computed eigenvalues, and these eigenvectors satisfy $B$-orthonormality, that is $q_i^T B q_j = \delta_{ij}$.

The user specifies the number of iteration vectors, $p (= NIV)$, and he/she has three options for the initial values of the iteration vectors, that is the start vectors:

1. Pseudo-random vectors.
2. Vectors generated (by the subroutine) on the basis of the diagonal elements of $A$ and $B$ (see page 41).
3. Start vectors may be input information.

Regardless of which option is used the user may also specify that a pseudo-random vector is to be included, as the last iteration vector, after each iteration cycle.

The only limitation put on the number of iteration vectors $NIV (= p)$ is that it must be greater than the number of requested eigenpairs $NEVAL (= m)$.

The user also controls the following parameters:

- The maximum number of iteration cycles (MAXIT).
- An indicator specifying which one of the two (input) matrices, $A$ or $B$, is most likely to be positive definite. This information is used by the subroutine to decide which subspace eigenproblem should be solved (the 'normal' one or the inverse).
- A vector operation code that makes it possible to speed up the iteration process by taking advantage of already 'converged vectors'.

The subroutines
• The tolerance value, \( \epsilon \), for the largest acceptable relative error in a computed eigenvalue, see Eq. (3.16).

**Subroutine LANCZ2 (SPRLAN)**

This subroutine computes the eigenvalues and, optionally, the corresponding eigenvectors by a truncated Lanczos method. The computational procedure is described in detail in Section 3.2. The reduced, tridiagonal eigenproblem, is solved by the implicit QL algorithm that was also used in subroutine HQLS (the solution is accomplished by subroutine QLS3D or QLS3DV).

The user controls the basic task of the subroutine by the values he or she specifies for the following three (input) parameters (integer variables):

- **NEVAL** - number \((= m)\) of requested eigenvalues.
- **NEVEC** - number \((= q)\) of requested eigenvectors, corresponding to the first **NEVEC** eigenvalues \((**NEVEC$$ \leq **NEVAL)\).
- **MAXLAN** - maximum number \((= p)\) of Lanczos steps (vectors) to be used (this number must not exceed the number of non-zero diagonal elements in \(B\)).

The process is halted when \(m\) 'good' eigenvalues (defined in Section 3.2, on page 54) have been determined or when \(p\) steps have been carried out, whichever comes first. If the latter comes first, that is if all **MAXLAN** steps have been carried out, the actual number of 'good' eigenvalues returned, \(m_j\), which is most likely less than the requested number \((m_j \leq m)\), is reported to the user. The actual number of eigenvectors returned, \(q_j\), which can never be larger than \(m_j\), is also reported back to the user.

LANCZ2 (SPRLAN) also returns \(p_j\) orthogonal Lanczos vectors \(V = [v_1, v_2, \ldots, v_{p_j}]\) or, optionally, approximations to \(q_j\) eigenvectors \(Q = [q_1, q_2, \ldots, q_{q_j}]\) corresponding to the first \(q_j\) eigenvalues computed. Here \(q_j \leq q \leq m_j \leq m \leq p_j \leq p\), where \(p\) is the maximum number of Lanczos steps to be performed and \(p_j\) is the actual number performed. Parameters \(m, q, p\) are input arguments, whereas \(m_j, q_j, p_j\) are all output information.

The eigenvectors, if computed, are \(B\)-orthonormal, that is \(q_j^T B q_j = \delta_{ij}\).

The user has four options for the start vector:

1. A pseudo-random vector.
2. The diagonal of the \(B\) matrix.
3. A vector of unit elements.
4. The start vector may be input information.
The user may specify one of five reorthogonalization procedures:
1. No reorthogonalization.
2. A ‘single pass’, full reorthogonalization, for which each new
   Lanczos vector is orthogonalized (by modified Gram-Schmidt)
   once with respect to all previous Lanczos vectors.
3. A ‘double pass’, full reorthogonalization, for which each new
   Lanczos vector is orthogonalized twice with respect to all previ­
   ous Lanczos vectors.
4. SIMON’s partial reorthogonalization, see page 50.
5. A simplified ‘half’ reorthogonalization process, see page 52.
The user also controls the following parameters:
- A parameter, KEX, specifying the frequency by which the
  ‘small’ tridiagonal eigenvalue problem is solved, and thus indi­
  rectly the stop criteria. This is the s-parameter described on page
  54.
- An indicator specifying which one of the two (input) matrices, A
  or B, should be decomposed into its Cholesky factor.
- The tolerance value, ε, for the largest acceptable relative error in
  a computed eigenvalue, see Eq. (3.36).

7.2 Subspace iteration - MSSIT

In this section we subject MSSIT to a series of tests, all of which are
concerned with free vibrations of 3D test frames defined in Tables
4.1 and 4.2. The purpose is to establish some information about the
accuracy and robustness of the subroutine and the computational
efforts involved. We also seek to provide some information/guide­
lines concerning some of the input parameters controlling the solu­
tion procedure.

Computations reported in this section are obtained by Computers I
and IV, and while some CPU-times will be given, we shall return to
the computational efficiency of the subroutine in a later section of
this chapter, where we compare it with the truncated Lanczos
method.

Unless otherwise stated, all computations are carried out for a con­
sistent mass matrix (B) and with pseudo-random start vectors.
PART II

Accuracy and robustness

We start with a small problem, for which we also have solutions obtained by GENHQL / GENJAC. Frame A-1, with \( M=N=1 \) and \( P=10 \) (\( n=240 \)), is analyzed with MSSIT using the following parameters:

\[
m = \text{NEVAL} = 30 \quad p = \text{NIV} = 45
\]

\[
\text{MAXIT} = 20 \quad \epsilon = 10^{-10}
\]

The solution was obtained in 3.41 seconds on Computer IV using 15 iterations. Some of the eigenvalues compare as follows with those obtained by GENHQL for the same problem:

<table>
<thead>
<tr>
<th>No.</th>
<th>Computed eigenvalues</th>
<th>Relative error</th>
<th>GENHQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.56160132567800D+01</td>
<td>3.7587D-16</td>
<td>7.56160132577016D+01</td>
</tr>
<tr>
<td>2</td>
<td>8.8094259625416D+01</td>
<td>1.2905D-15</td>
<td>8.80942596206315D+01</td>
</tr>
<tr>
<td>3</td>
<td>2.2263684586184D+02</td>
<td>4.4681D-15</td>
<td>2.22636845862693D+02</td>
</tr>
<tr>
<td>4</td>
<td>8.9588628915559D+02</td>
<td>3.8070D-16</td>
<td>8.95886289155486D+02</td>
</tr>
<tr>
<td>10</td>
<td>6.94916611006476D+03</td>
<td>2.6568D-14</td>
<td>6.9491661100694D+03</td>
</tr>
<tr>
<td>20</td>
<td>2.44521114733847D+04</td>
<td>6.2487D-15</td>
<td>2.44521114733812D+04</td>
</tr>
<tr>
<td>30</td>
<td>5.54536429206480D+04</td>
<td>2.7373D-11</td>
<td>5.54536429203047D+04</td>
</tr>
</tbody>
</table>

The first two columns apply to MSSIT. These results compare very well. It is in fact hard to say which are the more accurate; for the lower eigenvalues (1 through 4) MSSIT probably has the edge, number 20 is obtained by fairly well the same accuracy by both subroutines, whereas GENHQL probably finds the 30th with slightly higher accuracy (but there is very little in it).

Characteristics of the product \( Q_m^T B Q_m \):

<table>
<thead>
<tr>
<th>EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 30)</th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.0000000000000003E+00</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999950E-01</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>3.396241621267393E-14</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.0000000000000000E+00</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

show that orthogonality of the eigenvectors is well maintained. \( Q_m \) is the matrix of (\( m = 30 \)) eigenvectors, as returned by the subroutine (no ‘postprocessing’ has been applied).

In the above analysis the frame is supported at the base. If we remove the supports the matrices become singular (with a defect of 6, corresponding to the six rigid body modes). However, specifying a negative shift value, say \( \sigma = -1000 \), the subroutine should identify six zero eigenvalues.
NUMERICAL RESULTS

Using the same parameters as above, but with \( \sigma = -1000 \) and no supports, we find the first 10 eigenvalues to be:

<table>
<thead>
<tr>
<th>Computed eigenvalues</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  -6.48014975013211D-12</td>
<td>2.3874D-15</td>
</tr>
<tr>
<td>2  7.61701812734827D-12</td>
<td>0.0000D+00</td>
</tr>
<tr>
<td>3  3.87672116630711D-11</td>
<td>1.7053D-15</td>
</tr>
<tr>
<td>4  4.09272615797818D-11</td>
<td>2.3874D-15</td>
</tr>
<tr>
<td>5  5.40012479177676D-11</td>
<td>1.5916D-15</td>
</tr>
<tr>
<td>6  8.17408363218419D-11</td>
<td>1.3642D-15</td>
</tr>
<tr>
<td>7  7.50946287747924D+02</td>
<td>1.8180D-15</td>
</tr>
<tr>
<td>8  1.67255651134364D+03</td>
<td>6.8062D-16</td>
</tr>
<tr>
<td>9  1.84467896168425D+03</td>
<td>1.5986D-16</td>
</tr>
<tr>
<td>10 3.64413108247775D+03</td>
<td>1.1750D-15</td>
</tr>
</tbody>
</table>

For all practical purposes the first six eigenvalues are (numerically) zero. It should be noted that the relative errors apply to the eigenvalues as found by the iteration, that is to the values of \( \gamma \), see Eq. (1.61), before adjustments due to the shift value are made. The characteristics of the eigenvector product \( Q_m^T B Q_m \) is now:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element : 1.000000000000007E+00</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Smallest diagonal element : 9.999999999999946E-01</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Largest off-diag. element : 3.825498845397854E-15</td>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>Smallest off-diag. element : 5.963111948670274E-07</td>
<td>25</td>
<td>9</td>
</tr>
</tbody>
</table>

Again, orthogonality is very well preserved.

We next examine a symmetric problem, and this time we consider the larger problem posed by frame F-7 \((n = 3072)\). We request the 20 lowest eigenvalues and corresponding eigenvectors. The following parameters are used:

\[
m = \text{NEVAL} = 20 \quad p = \text{NIV} = 35 \quad \text{MAXIT} = 30 \quad \varepsilon = 10^{-10}
\]

The solution was obtained in 233 seconds on Computer IV using 24 iterations. The 10 first eigenvalues and their relative errors are as follows:

<table>
<thead>
<tr>
<th>Computed eigenvalues</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2.58575875835272D+02</td>
<td>4.3967D-16</td>
</tr>
<tr>
<td>2  2.58575875835328D+02</td>
<td>1.5388D-15</td>
</tr>
<tr>
<td>3  2.97483207732111D+02</td>
<td>5.7324D-16</td>
</tr>
<tr>
<td>4  2.39859267667285D+03</td>
<td>1.3271D-15</td>
</tr>
<tr>
<td>5  2.41133185442000D+03</td>
<td>1.5087D-15</td>
</tr>
<tr>
<td>6  2.41133185442006D+03</td>
<td>4.7147D-15</td>
</tr>
</tbody>
</table>
We see that the multiple eigenvalues are picked up with excellent agreement, and the characteristics of the matrix $Q^1BQ^m$:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 20)**

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000008E+00</td>
<td>14</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999973E-01</td>
<td>9</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>1.637624701101258E-15</td>
<td>11</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>1.43307367635869E-19</td>
<td>2</td>
</tr>
</tbody>
</table>

show that the eigenvectors satisfy the orthogonality requirement very well also in the case of multiple eigenvalues. As to the absolute accuracy of the eigenvalues we cannot, at present, make any claims. However, the same problem will also be analyzed with the Lanczos subroutine (LANCZ2), in the next section, and we reveal no secret by saying that the agreement is very good.

We notice that although $p - m = 15$ also in this case, as it was for the A-1 frame, we now require 24 iterations to obtain 20 eigenpairs compared to 15 iterations for 30 eigenpairs in the case of Frame A-1. This has to do with the spacing of the eigenvalues, a problem we shall return to.

Sometimes an 'almost' symmetric problem can be as awkward as a completely symmetric problem. Hence two almost identical problems, G-3 and H-3, are analyzed ($n = 1620$). The only difference is that the bay length, $b_y$, is 3.0001 m for the almost symmetric H-frame, whereas it is 3.0 for the symmetric G-frame. The following parameters are used:

$$m = \text{NEVAL} = 20 \quad p = \text{NIV} = 35$$

$$\text{MAXIT} = 30 \quad \varepsilon = 10^{-10}$$

The solution was obtained in about 15.5 seconds on Computer IV using 10 iterations, in both cases. The 10 first eigenvalues and their relative errors are as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>Computed eigenvalues</th>
<th>Relative error</th>
<th>Computed eigenvalues</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.97661115974824D+00</td>
<td>1.3401D-15</td>
<td>3.9765723127485D+00</td>
<td>2.6802D-15</td>
</tr>
<tr>
<td>2</td>
<td>3.97661115975831D+00</td>
<td>1.7868D-15</td>
<td>3.9767703088012D+00</td>
<td>2.2334D-15</td>
</tr>
<tr>
<td>3</td>
<td>2.7204274399617D+01</td>
<td>2.4813D-15</td>
<td>2.7203726455701D+01</td>
<td>1.0448D-15</td>
</tr>
<tr>
<td>4</td>
<td>7.50483659993519D+01</td>
<td>3.5978D-15</td>
<td>7.5047635604687D+01</td>
<td>3.5978D-15</td>
</tr>
<tr>
<td>5</td>
<td>7.50483659993578D+01</td>
<td>5.1126D-15</td>
<td>7.5048860455376D+01</td>
<td>1.5148D-15</td>
</tr>
<tr>
<td>6</td>
<td>2.45478578891952D+02</td>
<td>1.9683D-15</td>
<td>2.4547364611717D+02</td>
<td>4.8629D-15</td>
</tr>
<tr>
<td>7</td>
<td>3.39281695604393D+02</td>
<td>4.0209D-15</td>
<td>3.3928839229836D+02</td>
<td>6.7015D-16</td>
</tr>
</tbody>
</table>
NUMERICAL RESULTS

<table>
<thead>
<tr>
<th></th>
<th>8 3.39291695604398D+02</th>
<th>2.6806D-15</th>
<th>3.3928755327369D+02</th>
<th>2.8481D-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8.5432717180108D+02</td>
<td>6.6345D-16</td>
<td>6.85418458907745D+02</td>
<td>2.3221D-15</td>
</tr>
<tr>
<td>10</td>
<td>7.89356553623595D+02</td>
<td>4.3207D-16</td>
<td>7.89345039850765D+02</td>
<td>6.4812D-15</td>
</tr>
</tbody>
</table>

The two columns to the left apply to the symmetric G-3 frame, and the two to the right to the almost symmetric H-3 frame. The eigenvectors satisfy the orthogonality requirements with the same excellent accuracy in both cases. Hence, 'almost symmetry' poses no apparent problems for MSSIT.

Relative error tolerance, $\varepsilon$

We finally report on some analyses aimed at investigating the importance of the tolerance parameter $\varepsilon$. The problem is the symmetric G-3 frame ($n = 1620$). We use the same parameters as above for NEVAL ($-20$), NIV ($-35$) and MAXIT ($-30$), but for the error tolerance parameter $\varepsilon$ we now carry out separate analyses for four values: $10^{-6}$, $10^{-8}$, $10^{-10}$ and $10^{-12}$.

Eigenvalues (and eigenvectors) meeting these error tolerances are obtained in 10.8 seconds (and 7 iterations), 14.0 secs. (9), 15.5 secs. (10) and 18.5 secs. (12), respectively, using Computer IV. Three typical eigenvalues are as follows:

<table>
<thead>
<tr>
<th>EPS</th>
<th>1st eigenvalue</th>
<th>10th eigenvalue</th>
<th>20th eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>10. 0D-06</td>
<td>3.97661115974823D+00</td>
<td>7.89356553623595D+02</td>
<td>3.31637153385428D+03</td>
</tr>
<tr>
<td>10. 0D-08</td>
<td>3.97661115974823D+00</td>
<td>7.89356553623595D+02</td>
<td>3.31637153385428D+03</td>
</tr>
<tr>
<td>10. 0D-10</td>
<td>3.97661115974823D+00</td>
<td>7.89356553623595D+02</td>
<td>3.31637153385428D+03</td>
</tr>
<tr>
<td>10. 0D-12</td>
<td>3.97661115974823D+00</td>
<td>7.89356553623595D+02</td>
<td>3.31637153385428D+03</td>
</tr>
</tbody>
</table>

We see that the 1st (lowest) and the 10th eigenvalues are almost the same to 15 digits, for all values of $\varepsilon$. Since all eigenvalues have to meet the specified error tolerance, the first (lowest) eigenvalues will continue to improve as the iterations are carried out, and the error tolerance indicates the error level only for the last (highest) eigenvalue(s). The matrix product $Q_m^T B Q_m$ is,

in the case of $\varepsilon = 10^{-6}$:

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 20)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000034E+00</td>
<td>19</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999967E-01</td>
<td>12</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>7.956980072661851E-15</td>
<td>20</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>4.213575322283868E-19</td>
<td>2</td>
</tr>
</tbody>
</table>

and in the case of $\varepsilon = 10^{-12}$:

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 20)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000064E+00</td>
<td>20</td>
</tr>
</tbody>
</table>
In this particular case orthogonality seems to be as well preserved for the largest tolerance value as for the lowest!

Are these results representative? In order to answer that question we need more data, and we supplement the results obtained for the slender and symmetric G-3 frame with similar results for a ‘stocky’, non-symmetric Frame C-5 \((n=1296)\). We use the same parameters as above for NEVAL (=20), NIV (=35) and MAXIT (=30), and we carry out separate analyses for the same four values of \(\varepsilon\).

This problem, which has more closely spaced eigenvalues, requires more iteration cycles. Eigenvalues (and eigenvectors) that meet tolerance values of \(10^{-6}\), \(10^{-8}\), \(10^{-10}\) and \(10^{-12}\) are obtained in 27.4 seconds (and 11 iterations), 36.6 secs (15), 45.8 secs (19) and 52.8 secs (22 iterations), respectively, using Computer IV. Three typical eigenvalues are as follows:

<table>
<thead>
<tr>
<th>(\varepsilon)</th>
<th>1st eigenvalue</th>
<th>10th eigenvalue</th>
<th>20th eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10.0D-06)</td>
<td>3.05491588391497D+02</td>
<td>5.67440123903780D+03</td>
<td>1.35118185785055D+04</td>
</tr>
<tr>
<td>(10.0D-08)</td>
<td>3.05491588391497D+02</td>
<td>5.67440123903772D+03</td>
<td>1.35118145162149D+04</td>
</tr>
<tr>
<td>(10.0D-10)</td>
<td>3.05491588391497D+02</td>
<td>5.67440123903776D+03</td>
<td>1.35118144890407D+04</td>
</tr>
<tr>
<td>(10.0D-12)</td>
<td>3.05491588391497D+02</td>
<td>5.67440123903775D+03</td>
<td>1.35118144888488D+04</td>
</tr>
</tbody>
</table>

This is fairly well the same pattern as we found above, and in this case also the characteristics of the matrix product \(Q^T B Q_m\) show little if any difference for the different \(\varepsilon\)-values. Hence we have two somewhat different problems that seem to indicate much the same effect of the tolerance parameter \(\varepsilon\). This clearly does not provide enough ‘evidence’ for far reaching conclusions, but it seems fair to state that the computational effort is significantly increased by decreasing the error tolerance, and \(\varepsilon = 10^{-10}\) seems to be unnecessarily severe. \(10^{-8}\) or even \(10^{-6}\) will probably be sufficient in most cases.

**Computational effort**

We have already seen how the tolerance parameter \(\varepsilon\) has a significant influence on the computational effort. The form of (mass) matrix \(B\), skyline (sparse) or diagonal, will also influence the computational effort, as will the number and initial (start) values of the iterating vectors. The latter effect will be dealt with in the next subsection. In this subsection we will investigate the influence of the \(B\) matrix, and we will also take a closer look at which operations are responsible for the use of CPU-time.
The effect of $B$, on both accuracy and efficiency, is examined by analyzing Frames A-5 and B-5 ($n = 2160$). These frames are identical except for the mass model, which is consistent for A-5 and lumped for B-5. Both frames are analyzed with three different sets of parameters:

1. $m = \text{NEVAL} = 5$, $p = \text{NIV} = 15$
2. $m = \text{NEVAL} = 20$, $p = \text{NIV} = 35$
3. $m = \text{NEVAL} = 50$, $p = \text{NIV} = 75$

In all cases: $\text{MAXIT} = 30$ and $e = 10^{-10}$.

Table 7.1 gives the measured CPU-times. It is seen that for this particular problem there is on average about a 30 per cent time penalty for using consistent mass. This number will clearly be problem dependent, and Table 7.1 should only be taken as an indication.

Some of the eigenvalues computed (with $m = 50$) are as follows for the two mass models:

<table>
<thead>
<tr>
<th>No.</th>
<th>Consistent mass</th>
<th>Lumped mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.05320388665479D+02</td>
<td>1.05268863216627D+02</td>
</tr>
<tr>
<td>2</td>
<td>1.26741339932493D+02</td>
<td>1.26661734084333D+02</td>
</tr>
<tr>
<td>3</td>
<td>1.47801242482183D+02</td>
<td>1.44092641238465D+02</td>
</tr>
<tr>
<td>4</td>
<td>9.85500595284278D+02</td>
<td>9.81686971732630D+02</td>
</tr>
<tr>
<td>5</td>
<td>1.18190027292134D+03</td>
<td>1.17672451560565D+03</td>
</tr>
<tr>
<td>6</td>
<td>1.35940056277733D+03</td>
<td>1.32312960341681D+03</td>
</tr>
<tr>
<td>7</td>
<td>1.73952991470177D+03</td>
<td>1.72736281503333D+03</td>
</tr>
<tr>
<td>8</td>
<td>2.23346278777016D+03</td>
<td>2.10152060074350D+03</td>
</tr>
<tr>
<td>9</td>
<td>2.71831527148126D+03</td>
<td>2.6871713729197D+03</td>
</tr>
<tr>
<td>10</td>
<td>2.98456261807276D+03</td>
<td>2.9567279461798D+03</td>
</tr>
<tr>
<td>25</td>
<td>9.08148103390571D+03</td>
<td>8.55114231439435D+03</td>
</tr>
<tr>
<td>50</td>
<td>1.82006344957497D+04</td>
<td>1.62564943381821D+04</td>
</tr>
</tbody>
</table>

As could be expected the lumped mass model determines the lowest eigenvalues with quite good accuracy. But the higher eigenvalues
are not well predicted by a lumped mass model, it may be added that the lumped mass model is fairly crude, for the discretization used here.

Next we examine how the computer time is spent in the subroutine. Three main operations are recognized: factorization and substitution, matrix multiplication and solution of the small (subspace) eigenproblem. CPU-times are measured (and accumulated) for these three 'activities'. The total CPU-time is the sum of the times recorded for these three main operations; this means that some minor operations (never exceeding more than one per cent of the total time) are included in the matrix multiplication time.

Three frames are considered, A-2 ($n = 540$), A-4 ($n = 1500$) and A-6 ($n = 2940$), and for each frame computations are carried out for four different sets of parameters:

$$m = q / p = 5 / 15, \ 20 / 35, \ 50 / 75, \ 100 / 150.$$ 

In all cases: pseudo-random start vectors, consistent mass model and error tolerance $\varepsilon = 10^{-10}$. The results are shown in Figures 7.1, 7.2 and 7.3. The total CPU-times (as recorded by Computer I) are
Figure 7.2  MSSIT - relative computational effort for Frame A-4 \((n = 1500)\)

Figure 7.3  MSSIT - relative computational effort for Frame A-6 \((n = 2940)\)
shown at the top of each column indicating the relative distribution of computer time between the three main activities. We see that there is a shift in relative importance with both the dimension ($n$) of the problem and also with the number of eigenvalues/eigenvectors requested. Perhaps the most interesting finding is the relative small portion of the time spent in solving the reduced (subspace) eigenproblem (by GENHQL), particularly for the larger problems requiring few eigenpairs.

**Iteration or trial vectors**

There is clearly a trade off between the number of iteration vectors, $p$ (= NIV) and the number of iterations ($r$) required to find the requested number of eigenpairs, $m$ (= NEVAL), to a specified error tolerance. Due to the convergence characteristics of the method, see page 39, $r$ will go down with increasing $p$. The number of iterations may also depend on the values of the start vectors.

We first investigate the influence of the number of iteration vectors, $p$. The yardstick is the total CPU-time required to obtain a satisfactory solution. In all cases we use pseudo-random start vectors, consistent mass and an error tolerance $\varepsilon$ of $10^{-10}$.

Our first problem is Frame A-6 ($n = 2940$) for which we only request 5 eigenpairs, that is $m$ (= NEVAL) = 5. Table 7.2 summarizes the results from 8 computations. The table clearly shows the relationship between $p$ and $r$. However, in this particular case there is, in terms of computing time, not a great deal in it.

**Table 7.2:** CPU-times, $t$ (in seconds) and number of iterations, $r$, as a function of the number of iteration vectors, $p$.

<table>
<thead>
<tr>
<th>$p$ =</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$ =</td>
<td>19</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>$t$ =</td>
<td>39,1</td>
<td>37,4</td>
<td>38,9</td>
<td>39,9</td>
<td>38,0</td>
<td>40,5</td>
<td>43,4</td>
<td>39,5</td>
</tr>
</tbody>
</table>

Next we consider Frame A-2 ($n = 540$) with a request for $m = 20$ eigenpairs. Figure 7.4 summarizes 16 computations. The height of the columns indicates the total CPU-time, and the number of iterations required ($r$) is given at the top of each column. A similar picture is shown in Figure 7.5 for Frame C-3 ($n = 384$) with a request for $m = 20$ eigenpairs. The main difference between the two frames is that while A-2 is a fairly slender frame, with well spaced eigenvalues (at the low end of the spectrum), C-3 is a ‘stocky’ frame with more closely spaced eigenvalues. This explains why the optimal
NUMERICAL RESULTS

Figure 7.4  MSSIT - CPU-times for Frame A-2 \((n = 540)\) with \(m = 20\)

Figure 7.5  MSSIT - CPU-times for Frame C-3 \((n = 384)\) with \(m = 20\)
value for $p$ is smaller for A-2 (31) than for C-3 (51, which gives a slightly lower computer time than $p = 43$).

We have also carried out some computations for the same two frames, A-2 and C-3, requesting $m = 50$ eigenpairs. The results are summarized in Tables 7.3 and 7.4. In this case there is little difference between the frames, the optimal value for $p$ seems to be in the 70 to 80 range for both frames. This indicates that the eigenvalues in this range are about equally spaced for the two frames.

It is clear from the results presented that the ‘optimal’ number of iteration vectors, $p$, is problem dependent, and it is very hard to come up with a simple and generally applicable recommendation. However, the much quoted value for $p$ given by Eq. (3.17), first suggested by Bathe and Wilson [4], seems to underestimate the value for $p$ in most cases, except perhaps for low values of $m$. We suggest the following formula be used for computing the number of iteration vectors:

$$p = m + \text{int} \{m^{0.5}\} + 3$$

Some example combinations of $m/p$ by this formula are:

$$\frac{1}{5}, \frac{5}{11}, \frac{10}{19}, \frac{50}{75} \text{ and } \frac{100}{142}$$

Including more trial vectors, as the above formula suggests, is of course bad news for storage requirements. For certain problems there may therefore be a trade-off between speed and storage, particularly for very large problems.
The other question concerning the iteration vectors is their initial value, that is the content of the start vectors. So far only pseudo-random vectors have been used. The other option implemented is to let the subroutine generate a set of start vectors based on the diagonal terms of A and B. A slightly modified version of the procedure suggested by BATHÉ and WILSON [4], described in Section 3.1 on page 41, has been adopted. The modifications are that the first vector in the trial set \( \mathbf{V}_1 \) is taken to be a vector whose elements are all equal to unity (instead of the diagonal elements of the mass matrix \( \mathbf{B} \)), and the last vector of \( \mathbf{V}_1 \) is a pseudo-random vector.

Some of the computations carried out for Frames A-2 and C-3, reported in Figures 7.4 and 7.5, were repeated with the optional start vectors generated on the basis of \( \operatorname{diag}(\mathbf{A}) \) and \( \operatorname{diag}(\mathbf{B}) \). The results are summarized in Table 7.5

**Table 7.5:** CPU-times (in seconds) and number of iterations, \( r \), as a function of the number of \( p \) and initial value of the iteration vectors. Computer IV

<table>
<thead>
<tr>
<th>Frame (( m = 20 ))</th>
<th>( p )</th>
<th>Pseudo-random start vectors</th>
<th>Start vectors based on ( \operatorname{diag}(\mathbf{A},\mathbf{B}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( r )</td>
<td>CPU-t</td>
</tr>
<tr>
<td>A-2</td>
<td>28</td>
<td>22</td>
<td>7,36</td>
</tr>
<tr>
<td>A-2</td>
<td>31</td>
<td>15</td>
<td>6,04</td>
</tr>
<tr>
<td>A-2</td>
<td>36</td>
<td>13</td>
<td>6,10</td>
</tr>
<tr>
<td>C-3</td>
<td>27</td>
<td>50</td>
<td>13,8</td>
</tr>
<tr>
<td>C-3</td>
<td>47</td>
<td>13</td>
<td>7,69</td>
</tr>
</tbody>
</table>

The results are surprisingly poor for the start vectors generated on the basis of the diagonal elements of \( \mathbf{A} \) and \( \mathbf{B} \). So poor in fact that they caused a recheck of the code. However, no mistakes were found. It is possible that this particular frame problem may be unfavorable to this procedure for selecting start vectors. Nevertheless, the problem is a real physical problem that any general procedure ought to be able to accommodate. On this basis the automatically generated start vectors, using the diagonals of \( \mathbf{A} \) and \( \mathbf{B} \), are not recommended, at least not as implemented in MSSIT. Pseudo-random start vectors have never failed, and they should always be the first choice.

The subroutine allows the user to specify that a pseudo-random vector be fed into the trial vectors after each iteration cycle, as the last vector in the set. Regardless of which start vectors or the number of
trial or iteration vectors \((p)\) are being used, there seems to be no significant effect of including this pseudo-random vector. However, since it does not seem to slow down the process it is recommended to include it.

**Speed up**

At the end of Section 3.1, see page 44, a computational economizer is mentioned. This consists of ‘freezing’ already converged eigenvectors. By not recomputing these vectors in subsequent iteration cycles, a straightforward operations count indicates that significant savings can be made. The question is: how much will accuracy suffer? In order to answer this question several computations were carried out for Frames A-4 and A-6, with and without the speed up option.

In all computations a consistent mass model, resulting in a ‘skyline’ (sparse) \(B\) matrix, was used, and unless otherwise stated an error tolerance of \(\varepsilon = 10^{-10}\) was specified. The results are summarized in Table 7.6. It should be noted that \(n = 1500\) for Frame A-4, and it is 2940 for Frame A-6.

<table>
<thead>
<tr>
<th>Frame</th>
<th>(m)</th>
<th>(p)</th>
<th>Standard procedure</th>
<th>Modified, ‘economized’ procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(r)</td>
<td>CPU-t</td>
</tr>
<tr>
<td>A-4</td>
<td>5</td>
<td>15</td>
<td>9</td>
<td>10,8</td>
</tr>
<tr>
<td>A-4</td>
<td>20</td>
<td>35</td>
<td>23</td>
<td>59,8</td>
</tr>
<tr>
<td>A-4</td>
<td>50</td>
<td>75</td>
<td>21</td>
<td>166</td>
</tr>
<tr>
<td>A-4</td>
<td>100</td>
<td>150</td>
<td>20</td>
<td>432</td>
</tr>
<tr>
<td>A-6</td>
<td>50</td>
<td>75</td>
<td>25</td>
<td>542</td>
</tr>
</tbody>
</table>

We see that for the larger problems there is a time saving of about 25%. As for the accuracy, the following selected numbers for Frame A-4 with \(m = 100\) show that, while most eigenvalues will be determined with an accuracy much higher than suggested by \(\varepsilon\) in the standard procedure (numbers to the left), the accuracy in the modified (‘economized’) procedure is fairly constant and in the neighborhood of \(\varepsilon\) for all, but the first few eigenvalues:
NUMERICAL RESULTS

<table>
<thead>
<tr>
<th>Computed eigenvalues</th>
<th>Relative error</th>
<th>Computed eigenvalues</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.04197535079838D+02</td>
<td>2.7277D-16</td>
<td>1.04197535079838D+02</td>
<td>2.1821D-15</td>
</tr>
<tr>
<td>1.22390337476721D+02</td>
<td>0.0000D+00</td>
<td>1.22390337476721D+02</td>
<td>6.8505D-15</td>
</tr>
<tr>
<td>1.5181112377248D+02</td>
<td>7.4887D-15</td>
<td>1.5181112377248D+02</td>
<td>9.5481D-15</td>
</tr>
<tr>
<td>1.15600732826262D+03</td>
<td>0.0000D+00</td>
<td>1.15600732826262D+03</td>
<td>1.2981D-14</td>
</tr>
<tr>
<td>1.4039481233824D+03</td>
<td>7.4498D-15</td>
<td>1.4039481233824D+03</td>
<td>4.9557D-14</td>
</tr>
<tr>
<td>2.56547858118128D+03</td>
<td>1.4181D-15</td>
<td>2.56547858118129D+03</td>
<td>1.1612D-12</td>
</tr>
<tr>
<td>2.99224271386582D+03</td>
<td>4.4073D-15</td>
<td>2.99224271386585D+03</td>
<td>7.5852D-12</td>
</tr>
<tr>
<td>3.3019873954801D+03</td>
<td>1.1018D-15</td>
<td>3.3019873954808D+03</td>
<td>2.0699D-11</td>
</tr>
<tr>
<td>3.55023142566529D+03</td>
<td>4.2270D-15</td>
<td>3.55023142566550D+03</td>
<td>4.6133D-11</td>
</tr>
<tr>
<td>2.38210506372011D+04</td>
<td>1.0385D-14</td>
<td>2.38210506372357D+04</td>
<td>3.2761D-11</td>
</tr>
<tr>
<td>6.39984733726424D+04</td>
<td>5.9387D-11</td>
<td>6.39984733726439D+04</td>
<td>5.9245D-11</td>
</tr>
</tbody>
</table>

For the product $Q_m^T B Q_m$, the standard procedure gives the following characteristics:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000085E+00</td>
<td>64</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999992E-01</td>
<td>84</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>6.358921424021321E-14</td>
<td>86</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>0.000000000000000E+00</td>
<td>11</td>
</tr>
</tbody>
</table>

whereas the same numbers for the modified procedure are:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000052E+00</td>
<td>91</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999935E-01</td>
<td>85</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>5.765576868924591E-09</td>
<td>95</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>9.423241538204091E-20</td>
<td>2</td>
</tr>
</tbody>
</table>

Although the eigenvector orthogonality properties have suffered noticeably in the modified procedure, the eigenvectors are probably still acceptable, from a practical point of view. A tentative conclusion may be that the economized procedure, which cuts the computational costs significantly, does lead to reduced, but acceptable accuracy, provided a fairly strict error tolerance parameter, $\varepsilon$, is specified.

However, what happens if we lower the error tolerance requirement, that is, if we increase the value of $\varepsilon$? Let us reexamine Frame A-4 with $m = 50$ and $p = 75$ and with $\varepsilon = 10^{-7}$. The solution is obtained in $r = 15$ iterations using 116 seconds of CPU-time. This is less than both the standard (169) and the modified (129) procedures required for $\varepsilon = 10^{-10}$, see Table 7.6. And what is more is that about 35 of the lowest eigenvalues, out of the 50, are determined with Subspace iteration - MSSIT 109
higher accuracy than are the same eigenvalues using the modified procedure with $\varepsilon = 10^{-10}$. Also, the eigenvector product $Q_m' B Q_m$, the characteristics of which are:

<table>
<thead>
<tr>
<th>Extreme (Numerically) Elements of Matrix (Dim = 50)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>Largest diagonal element</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
</tr>
</tbody>
</table>

compare very well with the corresponding characteristics for the standard procedure with the much stronger requirement of $\varepsilon = 10^{-10}$. For the modified procedure these numbers are, however, much the same as those quoted for the $m = 100$ case above, and thus considerably less accurate.

Although some of the higher eigenvalues in the requested cluster may be slightly less accurate, overall accuracy, as well as speed, seem to be better served by increasing the error tolerance rather than ‘freezing’ eigenvectors corresponding to already converged eigenvalues. Hence, the modified (and faster) procedure is not recommended.

### 7.3 Truncated Lanczos - LANCZ2

In this section we repeat many of the tests of the previous section plus some new ones. The target is now subroutine LANCZ2. Again all tests are concerned with free vibrations of 3D test frames defined in Tables 4.1 and 4.2.

Computations reported in this section are obtained by Computers I, II and IV, and while some CPU-times will be given, we shall return to the computational efficiency of the subroutine in the next section of this chapter, where we compare it with subspace iteration.

Unless otherwise stated, all computations are carried out for

- a consistent mass matrix ($B$),
- with a pseudo-random start vector,
- with Cholesky factorization of matrix $A$, and
- with full, ‘single pass’ reorthogonalization.
NUMERICAL RESULTS

Accuracy and robustness

We start with the same problem as in the previous section, namely Frame A-1, with \( M=N=1 \) and \( P=10 \) (\( n=240 \)). The following parameters are used:

\[
\begin{align*}
  m &= \text{NEVAL} = q = \text{NEVEC} = 30 \\
  p &= \text{MAXLAN} = 75 \\
  s &= \text{KEX} = 2 \\
  \varepsilon &= 10^{-10}
\end{align*}
\]

The \( s \)-parameter is described on page 54 of Section 3.2.

The solution was obtained in 0.69 seconds on Computer II using 64 Lanczos step. Some of the eigenvalues compare as follows with those obtained by MSSIT for the same problem:

<table>
<thead>
<tr>
<th>No.</th>
<th>Computed ( \text{Re} )lative errors</th>
<th>Computed ( \text{Re} )lative errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.56160132567942\times10^04 + 01</td>
<td>1.1860E-15</td>
</tr>
<tr>
<td>2</td>
<td>8.80942596254239\times10^01 + 01</td>
<td>1.6810E-15</td>
</tr>
<tr>
<td>3</td>
<td>2.22636845618600\times10^02 + 02</td>
<td>5.7932E-16</td>
</tr>
<tr>
<td>4</td>
<td>8.95886289155910\times10^02 + 02</td>
<td>7.7706E-16</td>
</tr>
<tr>
<td>10</td>
<td>6.94916110064660\times10^03 + 03</td>
<td>1.0548E-14</td>
</tr>
<tr>
<td>20</td>
<td>2.44521147338614\times10^04 + 04</td>
<td>5.6833E-14</td>
</tr>
<tr>
<td>30</td>
<td>5.54536429203018\times10^04 + 04</td>
<td>1.2588E-13</td>
</tr>
</tbody>
</table>

The two columns to the left apply to LANCZ2, the two to the right to MSSIT. The numbers speak for themselves.

Characteristics of the product \( \mathbf{Q}_q^T \mathbf{B} \mathbf{Q}_q \):

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.0000000000000000E+00</td>
<td>27</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.99999999999981E-01</td>
<td>30</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>3.85535380746995E-14</td>
<td>25</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>1.81603863891322E-17</td>
<td>29</td>
</tr>
</tbody>
</table>

show that orthogonality of the eigenvectors \( \mathbf{Q}_q \) (as returned from the subroutine) is well preserved.

In the above analysis the frame is supported at the base. As for MSSIT we remove the supports. The matrices become singular (with a defect of 6, corresponding to the six rigid body modes). However, specifying a negative shift value, say \( \sigma = -1000 \), the subroutine should identify six zero eigenvalues. Using the same parameters as above, but with \( \sigma = -1000 \) and no supports, we find the first 10 eigenvalues to be:

1. This computer is slower than Computer IV.
PART II

<table>
<thead>
<tr>
<th>Computed</th>
<th>Relative</th>
<th>Accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eigenvalues</td>
<td>errors</td>
</tr>
<tr>
<td>1</td>
<td>-1.284661266254261D-11</td>
<td>4.3368D-16</td>
</tr>
<tr>
<td>2</td>
<td>5.45698210637569D-12</td>
<td>4.3368D-16</td>
</tr>
<tr>
<td>3</td>
<td>2.10326497849897D-11</td>
<td>6.5052D-16</td>
</tr>
<tr>
<td>4</td>
<td>5.206857167650014D-11</td>
<td>4.3368D-16</td>
</tr>
<tr>
<td>5</td>
<td>6.548361852765083D-11</td>
<td>4.3368D-16</td>
</tr>
<tr>
<td>6</td>
<td>8.12860897095546D-11</td>
<td>6.5052D-16</td>
</tr>
<tr>
<td>7</td>
<td>7.509462877479332D+02</td>
<td>1.1390D-15</td>
</tr>
<tr>
<td>8</td>
<td>1.672558511343627D+03</td>
<td>2.7527D-15</td>
</tr>
<tr>
<td>9</td>
<td>1.84467861684263D+03</td>
<td>1.8505D-15</td>
</tr>
<tr>
<td>10</td>
<td>3.644131082477754D+03</td>
<td>1.2588D-15</td>
</tr>
</tbody>
</table>

For all practical purposes the first six eigenvalues are (numerically) zero. It should be noted that the relative errors apply to the eigenvalues as found by the truncated Lanczos process, that is to the values of \( \gamma \), see Eq. (1.61), before adjustments due to the shift value are made. The number of the Lanczos step at which the individual eigenvalues were accepted, as 'good' eigenvalues, is also recorded in the above listing. A total of 66 steps were required to find the requested number of 30 eigenvalues (in fact 32 'good' and 33 'acceptable' eigenvalues were found). The last four eigenvalues in the above listing, from number 7 to number 10, compare very well with the same eigenvalues found by MSSIT, see page 97.

The characteristics of the eigenvector product \( Q^T B Q \) is now:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 30)**

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000000E+00</td>
<td>18</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.99999999999993E-01</td>
<td>22</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>4.528477070450148E-15</td>
<td>25</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>1.599198204416119E-18</td>
<td>25</td>
</tr>
</tbody>
</table>

It seems fair to conclude that this problem is as well handled by LANCZ2 as it was by MSSIT.

We continue to reanalyze the problems of the previous section, and turn now to the symmetric problem posed by Frame F-7 \((n=3072)\). We request the 20 lowest eigenvalues and corresponding eigenvectors. The following parameters are used:

\[
m = \text{NEVAL} = q = \text{NEVEC} = 20 \\
p = \text{MAXLAN} = 60 \\
s = \text{KEX} = 2 \\
\epsilon = 10^{-10}
\]

The solution was obtained in 46.8 seconds on Computer II using 58 Lanczos steps. The requested eigenvalues compare as follows with those obtained by MSSIT for the same problem:
## NUMERICAL RESULTS

<table>
<thead>
<tr>
<th>No.</th>
<th>Computed eigenvalues</th>
<th>Relative errors</th>
<th>Computed eigenvalues</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.58575875835272D+02</td>
<td>4.3967D-16</td>
<td>2.58575875835266D+02</td>
<td>3.3549D-15</td>
</tr>
<tr>
<td>2</td>
<td>2.5857587583532D+02</td>
<td>1.5388D-15</td>
<td>2.58575875835325D+02</td>
<td>3.3642D-16</td>
</tr>
<tr>
<td>3</td>
<td>2.9748320773211D+02</td>
<td>5.7324D-16</td>
<td>2.97483207732102D+02</td>
<td>2.5803D-13</td>
</tr>
<tr>
<td>4</td>
<td>2.39859266767285D+03</td>
<td>1.3271D-15</td>
<td>2.39859266767283D+03</td>
<td>1.9504D-13</td>
</tr>
<tr>
<td>5</td>
<td>2.41133185442000D+03</td>
<td>1.5087D-15</td>
<td>2.41133185442000D+03</td>
<td>2.6144D-16</td>
</tr>
<tr>
<td>6</td>
<td>2.41133185442000D+03</td>
<td>4.7147D-15</td>
<td>2.41133185442000D+03</td>
<td>3.6601D-13</td>
</tr>
<tr>
<td>7</td>
<td>2.75053183189476D+03</td>
<td>3.9599D-16</td>
<td>2.75053183189476D+03</td>
<td>8.4964D-16</td>
</tr>
<tr>
<td>8</td>
<td>4.59120151987854D+03</td>
<td>1.9810D-16</td>
<td>4.59120151987851D+03</td>
<td>2.4889D-16</td>
</tr>
<tr>
<td>9</td>
<td>5.24682481602763D+03</td>
<td>2.9367D-15</td>
<td>5.24682481602763D+03</td>
<td>9.8922D-16</td>
</tr>
<tr>
<td>10</td>
<td>5.24682481602763D+03</td>
<td>2.2457D-15</td>
<td>5.24682481602763D+03</td>
<td>2.4250D-15</td>
</tr>
<tr>
<td>11</td>
<td>7.31730792691928D+03</td>
<td>2.4859D-16</td>
<td>7.31730792691928D+03</td>
<td>5.1567D-15</td>
</tr>
<tr>
<td>12</td>
<td>7.31730792691935D+03</td>
<td>3.7288D-16</td>
<td>7.31730792691935D+03</td>
<td>4.3634D-15</td>
</tr>
<tr>
<td>13</td>
<td>7.67262467675092D+03</td>
<td>1.8735D-15</td>
<td>7.67262467675092D+03</td>
<td>6.3160D-16</td>
</tr>
<tr>
<td>14</td>
<td>7.67262467675093D+03</td>
<td>2.3419D-16</td>
<td>7.67262467675093D+03</td>
<td>5.8949D-15</td>
</tr>
<tr>
<td>15</td>
<td>8.08810058866880D+03</td>
<td>2.6988D-15</td>
<td>8.08810058866880D+03</td>
<td>4.3846D-15</td>
</tr>
<tr>
<td>16</td>
<td>9.73747272629145D+03</td>
<td>6.7249D-15</td>
<td>9.73747272629145D+03</td>
<td>1.0953D-14</td>
</tr>
<tr>
<td>17</td>
<td>1.12426197001107D+04</td>
<td>5.5694D-14</td>
<td>1.12426197001107D+04</td>
<td>3.0365D-16</td>
</tr>
<tr>
<td>18</td>
<td>1.33053978031582D+04</td>
<td>1.3818D-11</td>
<td>1.33053978031649D+04</td>
<td>1.8032D-15</td>
</tr>
<tr>
<td>19</td>
<td>1.33053978031491D+04</td>
<td>4.6948D-11</td>
<td>1.33053978031642D+04</td>
<td>5.5656D-16</td>
</tr>
<tr>
<td>20</td>
<td>1.36890600667924D+04</td>
<td>3.9210D-11</td>
<td>1.36890600667924D+04</td>
<td>3.6142D-15</td>
</tr>
</tbody>
</table>

The eigenvalues to the left are obtained by MSSIT. The far right column indicates the Lanczos step at which the eigenvalues were accepted (as 'acceptable').

The numbers above illustrate both the pros and the cons of the truncated Lanczos method. The method is clearly able to pick up multiple eigenvalues with excellent accuracy, but it does not find them all.

We see that the 19th eigenvalue, which is a multiple of (and equal to) the 18th, is missing in the Lanczos 'column'. All 20 eigenvalues returned by LANCZ2 as 'good' eigenvalues are certainly true eigenvalues, which is also supported by the characteristics of the eigenvector product $Q^T BQ$.

### EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 20)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element:</td>
<td>1.0000000000000007E+00</td>
<td>19</td>
</tr>
<tr>
<td>Smallest diagonal element:</td>
<td>9.999999999999990E-01</td>
<td>13</td>
</tr>
<tr>
<td>Largest off-diag. element:</td>
<td>4.258624160778657E-15</td>
<td>11</td>
</tr>
<tr>
<td>Smallest off-diag. element:</td>
<td>3.482999479109833E-18</td>
<td>14</td>
</tr>
</tbody>
</table>

But they do not all appear in their correct position in an ordered sequence. Hence, the stop criteria, that requires all $m$ (= 20) eigenvalues to not only satisfy the specified (relative) error tolerance, but also to appear in the same position (in an ordered sequence) in two consecutive eigenvalue extractions (solution of the reduced, tridiagonal eigenproblem), does not quite work in this case. Increasing the value of the KEX-parameter does, unfortunately, not help. Even the maximum value, which has somewhat arbitrarily been set to 5, does...
not force enough Lanczos steps to be carried out for the 19th eigenvalue to emerge on the scene. In fact, this is the only problem analyzed, for which the stop criteria has not produced the correct results with a KEX-value of 5 or less.

The reason why we have a problem here is easy to see from the above printout of the eigenvalues. Whilst the 11th eigenvalue was accepted at step No. 38, its twin was slow in coming, and was not accepted before step No. 54. This is unusually slow; in most symmetric problems analyzed the multiples have tended to appear in anything from 5 to 10 Lanczos steps. And once they appear, they find their correct place in just a few steps.

So what can be done? We could of course increase the value of KEX. In our current problem we can see that the 19th eigenvalue is on its way up through the sequence for KEX = 5, and it is quite possible that a value of 6 would have been enough to force a sufficiently large number of Lanczos steps to be carried out for the missing eigenvalue to find its place. However, in most cases large values of KEX will lead to unnecessary over-kill, and a better solution seems to be to ask for a few more eigenvalues than what is really required. In our case, for instance, \( m = 24 \) and \( \text{KEX} = 2 \) will find the 19th eigenvalue amongst the 24 'good' eigenvalues found in 86 Lanczos steps. \( \text{NEVAL} (= m) = 23 \) and \( \text{KEX} = 3 \) will also find the missing twin. And in most practical applications, such as modal analysis, for instance, it would probably not be noticeable if one of the higher eigenmodes was replaced by its neighbor.

The 'almost' symmetric problem represented by Frame H-3 was also analyzed with LANCZ2. The same results as those reported for MSSIT were reproduced, and since no surprises were observed the problem does not require any further comments.

Next we turn our attention to the relative error tolerance parameter \( \varepsilon \), and again we examine Frames G-3 (slender and symmetric with \( n = 1620 \)) and C-5 ('stocky' and non-symmetric with \( n = 1296 \)) for four different values of \( \varepsilon \), namely \( 10^{-6}, 10^{-8}, 10^{-10} \) and \( 10^{-12} \). For the other parameters we use:

\[
\begin{align*}
m &= \text{NEVAL} = q = \text{NEVEC} = 20 \\
p &= \text{MAXLAN} = 75 \\
s &= \text{KEX} = 2
\end{align*}
\]

For Frame G-3 eigenvalues (and eigenvectors) meeting these error tolerances are obtained in 3.30 seconds (using 52 Lanczos steps), 3.40 (54), 3.40 (54) and 3.63 (56), respectively, using Computer IV. Three typical eigenvalues are as follows:
The numbers show that there is very little difference between the computations, both in effort and accuracy, and the influence of \( \varepsilon \) is clearly less for LANCZ2 than for MSSIT. For the truncated Lanczos method, both effort and accuracy depend on the number of Lanczos steps performed, and due to the implemented stop criteria this number is not very sensitive to the value of \( \varepsilon \).

It is worth noting that for this symmetric problem, which has 8 multiple eigenvalues amongst the lowest twenty (in fact all three eigenvalues listed above are "twins"), the correct twenty eigenvalues are determined in all four computations. The problem we encountered for the symmetric Frame F-7, described above, is absent here.

The product \( Q_q^T B Q_q \) is almost indistinguishable for the four values of \( \varepsilon \). Typically the characteristics of the matrix for \( \varepsilon = 10^{-6} \) is:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.000000000000209E+00</td>
<td>20</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999114E-01</td>
<td>16</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>9.781026603410061E-14</td>
<td>17</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>3.506716401632803E-19</td>
<td>20</td>
</tr>
</tbody>
</table>

It can be argued that this is not quite as good as the similar result obtained by MSSIT, see page 99, but there is very little in it.

For the "stocky" Frame C-5 with more closely spaced, but distinct eigenvalues, we find 20 eigenvalues/eigenvectors meeting the specified error tolerances in

5,32 seconds (using 52 Lanczos steps), 5,32 secs. (52), 5,44 secs. (54) and 5,71 secs. (56),

for \( \varepsilon = 10^{-6}, 10^{-8}, 10^{-10} \) and \( 10^{-12} \), respectively,

using Computer IV. Three typical eigenvalues are as follows:

These numbers reinforce the findings of Frame G-3 above, and the product \( Q_q^T B Q_q \) is, for all practical purposes equal to the identity matrix in all four computations.
Hence we conclude that the truncated Lanczos procedure, as implemented in LANCZ2, is relatively insensitive to the error tolerance parameter $\epsilon$. This statement should perhaps be qualified to values of parameter $s (= \text{KEX})$ larger than 1. Recommended value for $\epsilon$ is $10^{-10}$.

We have so far used a value of 2 for parameter $s (= \text{KEX})$ in most cases. This parameter will influence the number of Lanczos steps and thus both accuracy and computational effort. Setting $\text{KEX} = 1$ will result in more frequent solutions of the small, tridiagonal eigenproblem (see figure on page 54) and also to a somewhat less severe stop criteria.

Reanalyzing Frames G-3 and C-5 with the same parameters as we used above, and with $\epsilon = 10^{-10}$, but with $\text{KEX} = 1$, give us a satisfactory solution for G-3 in 3,30 seconds (Computer IV) using 52 Lanczos steps. The same numbers for $\text{KEX} = 2$ were found above to be 3,40 and 54, respectively. For Frame C-5 we find 5,22 and 50 compared with 5,44 and 54 for $\text{KEX} = 2$.

Accuracy, for both eigenvalues and eigenvectors, is almost the same in both cases (with a slight advantage to $\text{KEX} = 2$). It is also interesting to note that we find the correct 20 eigenvalues/eigenvectors, even for the symmetric problem of G-3, using the minimum value of 1 for KEX. This is somewhat unusual. Most symmetric problems require a value of 2 or 3 for KEX in order to find all multiple eigenvalues included in the requested cluster. We have even seen (for Frame F-7 above) that a value of 5 for KEX was not sufficient for the subroutine to find the last multiple eigenvalue in the spectrum. Hence, in spite of the results reported for Frames G-3 and C-5, we recommend a value of 2 for KEX, for most cases, perhaps 3 if there is a high chance of problems with multiple eigenvalues being encountered.

Finally we address the problem of finding negative eigenvalues. With our test program that is only possible by specifying a positive shift value, $\sigma$, that is large enough to place the ‘new origin’ past the lowest eigenvalue of the basic (‘un-shifted’) problem (see figure on page 92). This means that the modified problem to be solved, defined by equation (3.42), will have one or more negative eigenvalues $\gamma_i$.

The problem is Frame A-6 ($n = 2940$). We first analyze it with the following parameters:

- $m = \text{NEVAL} = q = \text{NEVEC} = 100$
- $p = \text{MAXLAN} = 250$
- $s = \text{KEX} = 2$
- $\epsilon = 10^{-10}$ and $\sigma = 0.0$
**NUMERICAL RESULTS**

A satisfactory solution is obtained using 198 Lanczos steps. The 19th eigenvalue is about 5748. Next we reanalyze the problem with a shift value of $\sigma = 6000$. Hence the modified problem has 19 negative eigenvalues. We now seek the 20 eigenvalues, and corresponding eigenvectors, that are numerically closest to $\sigma$, that is we specify the following parameters:

$$m = \text{NEVAL} = q = \text{NEVEC} = 20$$
$$p = \text{MAXLAN} = 60$$
$$s = \text{KEX} = 2$$
$$\epsilon = 10^{-10}$$
$$\sigma = 6000.0$$

However, we have to make one more change. The modified $A$ matrix, $A_s$, see Eq. (3.43), is most likely indefinite, and we therefore base the solution on the matrix defined by Eq. (3.54) which involves the Cholesky factor of matrix $B$; since $B$ is a consistent mass matrix it is positive definite and thus has a Cholesky factor. We obtain a solution in 44 Lanczos steps. We compare the 20 eigenvalues obtained with $\sigma = 6000$, after the subroutine has added $\sigma$ to the computed $\gamma_i$ in order to obtain $\lambda_i$, with the 20 eigenvalues of the basic (‘unshifted’) problem that are (numerically) closest to the shift value of 6000:

<table>
<thead>
<tr>
<th>SHIFT = 0.0</th>
<th>SHIFT = 6000.</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 2.878546297682381D+03</td>
<td>1 5.7475370563038196D+03</td>
</tr>
<tr>
<td>11 2.979308596486972D+03</td>
<td>2 6.294842027852847D+03</td>
</tr>
<tr>
<td>12 3.652155671769744D+03</td>
<td>3 5.443076602682090D+03</td>
</tr>
<tr>
<td>13 3.752847837735443D+03</td>
<td>4 6.645253599456649D+03</td>
</tr>
<tr>
<td>14 3.829722253802078D+03</td>
<td>5 5.089886393564660D+03</td>
</tr>
<tr>
<td>15 4.347449409463674D+03</td>
<td>6 7.38468322334092D+03</td>
</tr>
<tr>
<td>16 4.590796787762333D+03</td>
<td>7 4.59079678776296D+03</td>
</tr>
<tr>
<td>17 5.089863935647303D+03</td>
<td>8 7.620483329637033D+03</td>
</tr>
<tr>
<td>18 5.443076602682061D+03</td>
<td>9 4.347449409463729D+03</td>
</tr>
<tr>
<td>19 5.747537056303827D+03</td>
<td>10 7.790075849418929D+03</td>
</tr>
<tr>
<td>20 6.294842027852847D+03</td>
<td>11 7.853256958977541D+03</td>
</tr>
<tr>
<td>21 6.645253599456649D+03</td>
<td>12 8.14492976661404D+03</td>
</tr>
<tr>
<td>22 7.38468322334092D+03</td>
<td>13 8.148365480804346D+03</td>
</tr>
<tr>
<td>23 7.620483329637033D+03</td>
<td>14 3.829722253802054D+03</td>
</tr>
<tr>
<td>24 7.790075849418854D+03</td>
<td>15 3.752847837735443D+03</td>
</tr>
<tr>
<td>25 7.853256958977541D+03</td>
<td>16 3.652155671769744D+03</td>
</tr>
<tr>
<td>26 8.14492976661404D+03</td>
<td>17 8.41595709812761D+03</td>
</tr>
<tr>
<td>27 8.148365480804346D+03</td>
<td>18 8.785911372397419D+03</td>
</tr>
<tr>
<td>28 8.41595709812761D+03</td>
<td>19 2.979308596486972D+03</td>
</tr>
<tr>
<td>29 8.785911372397419D+03</td>
<td>20 2.878546297682271D+03</td>
</tr>
</tbody>
</table>

The eigenvalues to the right, obtained by the last analysis, are ordered in a sequence corresponding to numerical ‘nearness’ to the shift value 6000; this is standard LANCZ2 ordering (subroutines are available for rearranging the ordering of both eigenvalues and eigenvectors). The patient reader will find that all 20 eigenvalues in the left hand column are also present in the right hand column.
The characteristics of the product $Q^T B Q_i$, for the last, 'shifted' problem show that eigenvector orthogonality is very well preserved:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Row</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal</td>
<td>$1.0000000000006E+00$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Smallest diagonal</td>
<td>$9.9999999999977E-01$</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Largest off-diag.</td>
<td>$1.121041329427488E-15$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Smallest off-diag.</td>
<td>$3.625301014248405E-18$</td>
<td>16</td>
<td>15</td>
</tr>
</tbody>
</table>

From this it seems fair to state that LANCZ2 has no problems with finding negative eigenvalues, and this last analysis also verify that a solution path based on the Cholesky factor of $B$ (instead of $A$) is a good alternative that should be used in situations like this. However, as seen from Eq. (3.64) we still require $A_o$ to be non-singular as it has be factored according to Eq. (3.57), and as will be shown in a later section a solution procedure based on the Cholesky factor of $A$ is cheaper than one based on $U_e$. For dynamic problems without shift ($\sigma = 0$) and for linearized bucking problems it is therefore recommended to factorize $A$ (the stiffness matrix).

It should be mentioned that the above, 'shifted' problem was also analyzed by MSSIT with the same excellent results.

Reorthogonalization

So far all computations with LANCZ2 have used full, 'one-pass' modified Gram-Schmidt reorthogonalization. In other words, each new Lanczos vector is made explicitly orthogonal to all preceding Lanczos vectors. However, as explained at the end of Section 7.1 (see page 95), and elaborated on in Section 3.2 (see page 50), the subroutine offers five options regarding reorthogonalization, controlled by parameter KORT:

- $0$ - no reorthogonalization
- $1$ - full, 'single-pass' reorthogonalization
- $2$ - full, 'double-pass' reorthogonalization
- $-1$ - partial reorthogonalization (due to SIMON)
- $-2$ - simple 'half' reorthogonalization

In this section we will test them all, and we choose Frame A-6 ($n = 2940$) as our test problem. All computations are carried out on Computer IV.

Our reference solution is obtained with full, single-pass reorthogonalization ($KORT = 1$) and with the following parameters:

\[
\begin{align*}
m &= \text{NEVAL} = q = \text{NEVEC} = 100 \\
p &= \text{MAXLAN} = 250
\end{align*}
\]
NUMERICAL RESULTS

\( s = \text{KEX} = 2 \quad \varepsilon = 10^{-10} \quad \sigma = 0. \)

A satisfactory solution is obtained in 71.6 seconds, 12.8 or about 18\% of which is spent in reorthogonalization, using 198 Lanczos steps (producing 101 'good' eigenvalues). The number of vector dot products performed during reorthogonalization, see Eq. (1.57), is a good measure of the work involved. In this case that number is 19503. We will return to some of the results of this computation in subsequent comparisons.

We first try a solution without any reorthogonalization. In anticipation of the problems we are likely to run into, we reduce the number of requested eigenvalues/eigenvectors and specify:

\[
m = \text{NEVAL} = q = \text{NEVEC} = 20
\]

\[
p = \text{MAXLAN} = 60
\]

The other parameters are unchanged. LANCZ2 returns with a warning saying that only 5 eigenvalues were accepted after 60 steps. And here are the 13 first eigenvalues determined, alongside the first 13 eigenvalues found in the reference solution (with KORT = 1):

<table>
<thead>
<tr>
<th>KORT = 1</th>
<th>KORT = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.060661895924409D+02</td>
<td>1.060661895924404D+02</td>
</tr>
<tr>
<td>1.296071934830195D+02</td>
<td>1.060661895924404D+02</td>
</tr>
<tr>
<td>1.451925764690007D+02</td>
<td>1.060661895924405D+02</td>
</tr>
<tr>
<td>9.900579913038688D+02</td>
<td>1.060661895924405D+02</td>
</tr>
<tr>
<td>1.199902161961889D+03</td>
<td>1.060661895924405D+02</td>
</tr>
<tr>
<td>1.268549190708086D+03</td>
<td>1.116687351598034D+02</td>
</tr>
<tr>
<td>1.33419330031542D+03</td>
<td>1.136687351598034D+02</td>
</tr>
<tr>
<td>1.625012722510369D+03</td>
<td>1.296071934830188D+02</td>
</tr>
<tr>
<td>2.250213950606870D+03</td>
<td>1.296071934830188D+02</td>
</tr>
<tr>
<td>2.87856297682381D+03</td>
<td>1.296071934830188D+02</td>
</tr>
<tr>
<td>2.979308596486972D+03</td>
<td>1.296071934830191D+02</td>
</tr>
<tr>
<td>3.652185567679744D+03</td>
<td>1.451925764690004D+02</td>
</tr>
<tr>
<td>3.752847837723544D+03</td>
<td>1.451925764690006D+02</td>
</tr>
</tbody>
</table>

The three columns to the right apply to the solution with no reorthogonalization, the last column indicating the Lanczos step at which the eigenvalues were accepted. It is interesting to note that the procedure finds the first eigenvalue five times (with excellent accuracy), and we also see that it finds the second and third eigenvalues, several times over, but not where they ought to be. From a practical point of view the solution is of course useless, as it stands, which is also obvious from the following characteristics of the eigenvector product \( Q^T \cdot B \cdot q_{ij} \):

EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 5)

<table>
<thead>
<tr>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element : 2.36987029057471E+00</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Smallest diagonal element : 2.48459434820743E-01</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truncated Lanczos - LANCZ2
What if we ask for fewer eigenvalues? Changing the first three parameters to

\[
m = \text{NEVAL} = q = \text{NEVEC} = 5
\]

\[
p = \text{MAXLAN} = 25
\]

does not help. We still find several ‘copies’ of the first eigenvalues. Reducing even further, to

\[
m = \text{NEVAL} = q = \text{NEVEC} = 3
\]

\[
p = \text{MAXLAN} = 15
\]

did produce a satisfactory solution, in 14 Lanczos steps. In spite of this we would state quite categorically that practical eigensolution by the truncated Lanczos should never be attempted without any form of reorthogonalization.

We now turn to the other extreme, that is a ‘double pass’, full reorthogonalization. This means that each Lanczos vector is orthogonalized twice with respect to all previous vectors, in two separate loops. With all other parameters being the same as for the reference solution, we obtain a satisfactory solution using exactly the same number of Lanczos steps (198).

The CPU-time spent in reorthogonalization doubles and now accounts for approximately 30% of the total time. As for accuracy, for both eigenvalues and eigenvectors, it is a toss-up between this and the reference solution. We will show some eigenvalues later on in this sub-section.

We have also examined a symmetric problem, Frame G-3 \((n = 1620)\), and with parameters

\[
m = \text{NEVAL} = q = \text{NEVEC} = 100
\]

\[
p = \text{MAXLAN} = 250
\]

we find that both \(\text{KORT} = 1\) and \(\text{KORT} = 2\) find a satisfactory solution in 244 Lanczos steps, and with indistinguishable accuracy. One might have suspected that \(\text{KORT} = 2\) would provide ‘too much’ orthogonality for this problem. After all, the process does depend on some numerical ‘noise’ for it to find multiple eigenvalues. However, the multiple eigenvalues are not slower in coming for \(\text{KORT} = 2\) than for \(\text{KORT} = 1\), if anything they are accepted at lower step numbers with the ‘double pass’ procedure. But there is very little in it, and in the end they both need 244 Lanczos steps in order to find 100 ‘good’ eigenvalues.

Next we try Simon’s partial reorthogonalization. With all other parameters the same as for the reference solution we find a satisfactory solution in the same number of Lanczos steps (198) with almost exactly half the reorthogonalization effort: 6.59 seconds, or
NUMERICAL RESULTS

about 10% of the total CPU-time, was spent and 9543 vector dot products were performed.

The accuracy for the eigenvalues is very good, but as will be seen, the eigenvectors are not quite as good as for the reference solution.

The following excerpt from the LANCZ2 printout shows how loss of orthogonality develops and how the partial reorthogonalization procedure works:

<table>
<thead>
<tr>
<th>Lanczos step</th>
<th>max. inner product</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6.5314E-15</td>
</tr>
<tr>
<td>4</td>
<td>4.5390E-15</td>
</tr>
<tr>
<td>5</td>
<td>1.2839E-14</td>
</tr>
<tr>
<td>6</td>
<td>1.6603E-14</td>
</tr>
<tr>
<td>7</td>
<td>8.4116E-14</td>
</tr>
<tr>
<td>8</td>
<td>2.0661E-12</td>
</tr>
<tr>
<td>9</td>
<td>1.1068E-10</td>
</tr>
<tr>
<td>10</td>
<td>6.0412E-09</td>
</tr>
<tr>
<td>11</td>
<td>2.5228E-07</td>
</tr>
</tbody>
</table>

Lanczos step 11 : reorthogonalize
Lanczos step 12 : reorthogonalize

From about the 60th step the pattern is: two steps without and two steps with reorthogonalization.

Finally we analyze our problem using the simple ‘half’ reorthogonalization procedure. With all other parameters the same as for the reference solution we again find a satisfactory solution in the same number of steps (198). The computational effort for the reorthogonalization process is almost the same as for the partial procedure, 9653 dot products are formed (compared to 9543). As for the accuracy we see from the values of three typical eigenvalues:

<table>
<thead>
<tr>
<th>KORT</th>
<th>1st eigenvalue</th>
<th>50th eigenvalue</th>
<th>100th eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.060661895924403D+02</td>
<td>1.4965545237852220D+04</td>
<td>3.506408566460306D+04</td>
</tr>
<tr>
<td>1</td>
<td>1.060661895924409D+02</td>
<td>1.4965545237852225D+04</td>
<td>3.506408566460315D+04</td>
</tr>
<tr>
<td>-1</td>
<td>1.060661895924407D+02</td>
<td>1.496554523785230D+04</td>
<td>3.506408566460458D+04</td>
</tr>
<tr>
<td>-2</td>
<td>1.060661895924407D+02</td>
<td>1.496554523785192D+04</td>
<td>3.506408566460412D+04</td>
</tr>
</tbody>
</table>

that there is very little in it.

However, if we examine the characteristics of the eigenvector product $Q_i^T B Q_q$ we see a more noticeable difference.
### KORT = 2:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th></th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000049E+00</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999624E-01</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>3.064074093463241E-14</td>
<td>100</td>
<td>99</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>2.710505431213761E-19</td>
<td>41</td>
<td>18</td>
</tr>
</tbody>
</table>

### EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)

<table>
<thead>
<tr>
<th></th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000049E+00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999624E-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>3.064074093463241E-14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>2.710505431213761E-19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### KORT = 1:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th></th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000037E+00</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999655E-01</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>2.470673240275508E-14</td>
<td>79</td>
<td>75</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>4.743384504624082E-19</td>
<td>14</td>
<td>7</td>
</tr>
</tbody>
</table>

### KORT = -1:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th></th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000079E+00</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999194E-01</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>5.547641145392900E-08</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>1.555152491158895E-18</td>
<td>89</td>
<td>45</td>
</tr>
</tbody>
</table>

### KORT = -2:

**EXTREME (NUMERICALLY) ELEMENTS OF MATRIX (DIM = 100)**

<table>
<thead>
<tr>
<th></th>
<th>VALUE</th>
<th>ROW</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest diagonal element</td>
<td>1.00000000000000070E+00</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>Smallest diagonal element</td>
<td>9.999999999999464E-01</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>Largest off-diag. element</td>
<td>2.577534625978673E-10</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Smallest off-diag. element</td>
<td>1.66018457618429E-19</td>
<td>46</td>
<td>34</td>
</tr>
</tbody>
</table>

We see that while there is little if any difference between the first two (full) procedures, there is a noticeable difference between the ‘full’ and the ‘reduced’ procedures. Of the reduced procedures the simple ‘half’ procedure is slightly better than the partial procedure. However, for practical purposes they are probably both well within required accuracy.

The ‘double pass’ full reorthogonalization seems to waste computer effort. The other three procedures all seem to be good candidates for the reorthogonalization job.
NUMERICAL RESULTS

Computational effort
We have already seen that while the reorthogonalization parameter (KORT) can have a noticeable effect on the total computing time, the error tolerance parameter (\( \varepsilon \)) seem to have a very moderate effect on the total CPU-time.

In this sub-section we will examine how the form of the (mass) matrix \( B \), matrix factorization and the type of start vector influence the computational effort. We are also interested in how the total CPU-time is spent during the computations, we identify five specific operations.

The effect of \( B \) is examined by analyzing Frames A-6 and B-6 (\( n = 2940 \)). These frames are identical except for the mass model, which is consistent for A-6 and lumped for B-6. Both frames are analyzed with three different sets of parameters \( m = q / p \). For all other parameters the standard values are used.

Table 7.7 gives the measured CPU-times. It is seen that for this particular problem there is on average about a 30 per cent time penalty for using consistent mass, which is about the same as subspace iteration for a similar problem. Again this should only be taken as an indication.

We next carry out similar computations, using Frame A-6, but this time we examine the effect of Cholesky factorizing matrix \( A \) versus matrix \( B \). Table 7.8 gives the measured CPU-times. As could be expected the effect is most noticeable for problems requesting few eigenvalues/eigenvectors. The number of Lanczos steps required is practically the same as is the accuracy of the solutions, for both eigenvalues and eigenvectors. Again, the numbers of Table 7.8 should only be taken as indications; the effect is no doubt problem dependent.

LANCZ2 offers several start vector options. So far we have only used a pseudo-random start vector. In addition we may specify a
PART II

Table 7.8: CPU-times for finding eigenvalues and eigenvectors of Frame A-6 \( (n = 2940) \)

<table>
<thead>
<tr>
<th>Parameters ( m = q / p )</th>
<th>CPU-time [seconds] - Computer IV</th>
<th>Time ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Factorizing A</td>
<td>Factorizing B</td>
</tr>
<tr>
<td>10 / 35</td>
<td>13.8</td>
<td>19.7</td>
</tr>
<tr>
<td>50 / 150</td>
<td>42.1</td>
<td>50.0</td>
</tr>
<tr>
<td>100 / 225</td>
<td>71.6</td>
<td>83.6</td>
</tr>
</tbody>
</table>

start vector consisting of the diagonal elements of \( B \) and also a start vector consisting of only unit elements. We have analyzed Frame A-6 with the same set of parameters \( m, q \) and \( p \) as in Tables 7.7 and 7.8 with these three different start vector options, and the result is that their performance is almost identical. If anything the vector of unit elements may have the edge, but there is very little in it. It seems fair to state that the start vector is of minor importance. However, and in spite of the above statement, we recommend a pseudo-random start vector, it has never failed. The subroutine also has an option for premultiplying the start vector by the \( H \) matrix. This, however, is a waste of time, and it is not recommended.

Finally we examine how the computer time is spent in the subroutine. Five main operations are recognized:

- factorization and substitution,
- matrix multiplication,
- reorthogonalization,
- solution of the small, tridiagonal eigenproblem, and
- finding the eigenvectors.

CPU-times are measured (and accumulated) for these five ‘activities’. The total CPU-time is the sum of the times recorded for these five main operations; this means that some minor operations (never exceeding more than one per cent of the total time) are included in the matrix multiplication time.

Four frames are considered, A-2 \( (n = 540) \), A-4 \( (n = 1500) \), A-6 \( (n = 2940) \) and C-9 \( (n = 6006) \), and for each frame computations are carried out for four different sets of parameters:

\[
m = q / p = 5 / 20, \ 20 / 60, \ 50 / 125, \ 100 / 200.
\]

In all cases: pseudo-random start vector, consistent mass model, full, ‘single pass’ reorthogonalization, error tolerance \( e = 10^{-10} \), \( KEX = 2 \) and Cholesky factorization of matrix \( A \). The results, which are obtained by Computers I and IV, are shown in Figures 7.6, 7.7, 7.8 and 7.9.
Figure 7.6 LANCZ2 - relative computational effort for frame A-2 \( (n = 540) \)

Figure 7.7 LANCZ2 - relative computational effort for frame A-4 \( (n = 1500) \)
PART II

Figure 7.8 LANCZ2 - relative computational effort for frame A-6 \((n = 2940)\)

Figure 7.9 LANCZ2 - relative computational effort for frame C-9 \((n = 6006)\)

126 THE GENERALIZED EIGENPROBLEM - Large and sparse matrices
NUMERICAL RESULTS

We see that the size of the problem, both in terms of matrix dimension ($n$) and perhaps even more in terms of the number of eigenvalues/eigenvectors requested, has a significant effect on how the total CPU-time breaks down on the various activities. On the whole we see that the basic operations of matrix multiplication (mainly matrix vector products) and matrix factorization and substitution are responsible for the bulk of the computer effort.

It is also worth noting that the small, tridiagonal eigenproblem accounts for very little of the total computational effort, unless a large number of eigenvalues/eigenvectors, relative to the dimension of the problem, is requested. Reorthogonalization, which is sometimes portrayed as a significant drawback of the method, seldom accounts for more than 20% of the total effort (usually a good deal less), even with full reorthogonalization. For all problems considered here (Figures 7.6 to 7.9), the reorthogonalization time could effectively have been halved, using either SIMON’s partial or the simple ‘half’ reorthogonalization procedure, without serious loss of accuracy.

### 7.4 Comparison between Lanczos and subspace iteration

We have now demonstrated that both methods are robust and reliable, and we have given some recommendations as to the most effective use of the methods per se. From some of the results presented in the previous sections it is also possible to make comparisons concerning computational efficiency. A more systematic comparison, however, is the subject of this section.

We are primarily interested in computational efficiency, but storage requirements will also be touched upon.

We define the following time ratio

$$\alpha_1 = \frac{\text{total CPU-time spent in MSSIT}}{\text{total CPU-time spent in LANCZ2}}$$

The time ratio $\alpha_1$ is, based on a series of computer runs for six frames of type A, tabulated in Table 7.9. For each frame up to four different numbers of eigenpairs ($m = q$) were requested. For two of the frames, A-2 and A-4, a few runs were also made with a lumped mass model, that is with frames B-2 and B-4.

For all computations:

- tolerance value for the relative eigenvalue error: $\varepsilon = 10^{-10}$,

and

- pseudo random start vectors.
Table 7.9: Time ratio $\alpha_1 = (\text{CPU-time in MSSIT}) / (\text{CPU-time in LANCZ2})$ for some unsymmetric frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>$n$</th>
<th>Time ratio $\alpha_1$ - Computer I $m = q = 5$</th>
<th>$m = q = 20$</th>
<th>$m = q = 50$</th>
<th>$m = q = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>240</td>
<td>4.24</td>
<td>6.26</td>
<td>8.52</td>
<td>16.41</td>
</tr>
<tr>
<td>A-2</td>
<td>540</td>
<td>3.84</td>
<td>6.89</td>
<td>11.49</td>
<td>17.95</td>
</tr>
<tr>
<td>B-2</td>
<td></td>
<td></td>
<td>6.25</td>
<td></td>
<td>17.63</td>
</tr>
<tr>
<td>A-3</td>
<td>960</td>
<td>4.56</td>
<td>9.64</td>
<td>15.24</td>
<td>21.81</td>
</tr>
<tr>
<td>A-4</td>
<td>1500</td>
<td>4.34</td>
<td>10.64</td>
<td>11.56</td>
<td>16.47</td>
</tr>
<tr>
<td>B-4</td>
<td></td>
<td></td>
<td>9.69</td>
<td></td>
<td>14.04</td>
</tr>
<tr>
<td>A-5</td>
<td>2160</td>
<td>4.10</td>
<td>8.65</td>
<td>12.42</td>
<td>18.30</td>
</tr>
<tr>
<td>A-6</td>
<td>2940</td>
<td>3.69</td>
<td>9.42</td>
<td>12.17</td>
<td>17.93</td>
</tr>
<tr>
<td>'Average' ratio$^a$</td>
<td>4.10</td>
<td>9.05</td>
<td>12.58</td>
<td></td>
<td>18.49</td>
</tr>
</tbody>
</table>

$^a$ Excluding A-1, B-2 and B-4.

For MSSIT (subspace iteration) the number of trial (iteration) vectors, $p$, are:

- 15 (5), 35 (20), 75 (50) and 150 (100)

Numbers in parentheses are the corresponding number of eigenpairs ($m = q$) requested. The number of iterations vary from 6 to 25.

For LANCZ2 (truncated Lanczos) the following parameters were specified in all cases:

- **KPD = 1**: Cholesky factorization of $A$.
- **KORT = 1**: Full (‘one pass’) reorthogonalization.
- **KEX = 2**: Stop criteria parameter.

Table 7.9 is obtained using Computer I. Some typical computing times for MSSIT are:

- $A-6 / m = q = 5$: 78.0 seconds
- $A-6 / m = q = 100$: 2490 seconds

Table 7.9 indicates a clear tendency: the more eigenpairs we require, the higher is the time ratio $\alpha_1$. Or in other words, the more eigenpairs we require, the more efficient becomes the truncated Lanczos algorithm relative to subspace iteration. This tendency is almost independent of the dimension (size) of the matrices involved, and we have included a row of average time ratios in Table 7.9, excluding the smallest problem (A-1) and the lumped
NUMERICAL RESULTS

mass models (B-2 and B-4). Figure 7.10, which shows the relative (and normalized) computational effort of the two subroutines, is prepared on the basis of the average time ratios in Table 7.9.

Figure 7.10 Relative computational effort of MSSIT and LANCZ2 as a function of the number of requested eigenvalues/eigenvectors (m/q); based on average time ratios for five test frames

These results, which are very much in favor of the truncated Lanczos algorithm, support the findings of many other investigators. One can of course argue that the subspace iteration routine, MSSIT, has not been executed with the most "optimal" set of parameters, and that it therefore has a speed up potential. True, but the same goes for the truncated Lanczos routine, LANCZ2, perhaps even more so. It should also be kept in mind that if only eigenvalues are required, LANCZ2 will be even more superior. MSSIT will find the eigenvectors regardless of whether or not they are required, whereas this is a separate operation in LANCZ2, an operation that can be quite costly, see Figures 7.6 to 7.9.

For most of the problems used to derive Table 7.9 and Figure 7.10 the eigenvalues are closely spaced, particularly the higher ones. This tends to favor the Lanczos algorithm, whereas it penalizes the subspace iteration algorithm. However, the problems are fairly typical structural engineering problems, and as such they are believed to provide representative results. It should also be mentioned that no dramatic effects have been found, on computer times, as a result
of the different eigenvalue spacing found in tall, slender frames and low, 'stocky' frames.

It seems fair to state that truncated Lanczos is far superior to subspace iteration, in terms of computational efficiency. This statement is believed to be equally valid for the various speed up modifications that can be applied to both algorithms. Of the various suggestions put forward in the literature it appears that the Lanczos algorithm has as much, if not more, going for it in terms of speed up potential, than has the subspace iteration technique.

Storage requirement is measured in terms of words, the storage unit holding an integer number and a 'normal' floating point number. All numerical operations reported in this investigation are carried out in so-called double precision, and a 'double precision' floating point number occupies two (consecutive) words of storage. The test program used does not report on the storage requirement of the eigensolution per se. However, all array information, both integer and floating point information, is stored in a common 'working array' (through some clever use of equivalencing storage space), and the program reports on the total number of words used in this working array. Since everything outside the eigensolution routines is the same, regardless of which eigensolver is used, this total storage requirement is a good indication also for the eigensolvers. Table 7.10 shows the total storage requirement for the test problems reported in Table 7.9; numbers in italic applies to LANCZ2.

Table 7.10: Total storage requirement, in terms of words, for MSSIT (regular digits) and LANCZ2 (italicized digits)

<table>
<thead>
<tr>
<th>Frame</th>
<th>$n$</th>
<th>$m = q = 5$</th>
<th>$m = q = 20$</th>
<th>$m = q = 50$</th>
<th>$m = q = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>240</td>
<td>44000</td>
<td>69320</td>
<td>134360</td>
<td>308180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39438</td>
<td>65838</td>
<td>122388</td>
<td>208638</td>
</tr>
<tr>
<td>A-2</td>
<td>540</td>
<td>152254</td>
<td>201574</td>
<td>314614</td>
<td>578314</td>
</tr>
<tr>
<td></td>
<td></td>
<td>142292</td>
<td>192692</td>
<td>288242</td>
<td>419492</td>
</tr>
<tr>
<td>A-3</td>
<td>960</td>
<td>396416</td>
<td>479336</td>
<td>659576</td>
<td>1049276</td>
</tr>
<tr>
<td></td>
<td></td>
<td>378894</td>
<td>462894</td>
<td>613044</td>
<td>862894</td>
</tr>
<tr>
<td>A-4</td>
<td>1500</td>
<td>850646</td>
<td>976766</td>
<td>1243406</td>
<td>1795106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>823404</td>
<td>950604</td>
<td>1170954</td>
<td>1446204</td>
</tr>
<tr>
<td>A-5</td>
<td>2160</td>
<td>1600048</td>
<td>1778968</td>
<td>2151208</td>
<td>2900908</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1560926</td>
<td>1740926</td>
<td>2047076</td>
<td>2524926</td>
</tr>
<tr>
<td>A-6</td>
<td>2940</td>
<td>2699630</td>
<td>2940950</td>
<td>3437990</td>
<td>4421690</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2676418</td>
<td>2950468</td>
<td>3328468</td>
<td>3956418</td>
</tr>
</tbody>
</table>
NUMERICAL RESULTS

In all but one case (A-6 / m=q=20) the storage requirement for LANCZ2 is less than that of MSSIT, in some cases the difference is quite significant. Without putting too much emphasis on these numbers, it seems fair to say that LANCZ2 has the edge also in terms of storage requirement.

7.5 Storage arrangement - skyline versus sparse

A fairly recent SAM library development is the inclusion of a ‘package’ for assembly and solution of symmetric, finite element systems by ‘true’ sparse matrix methods [18]. The sparse techniques used are those developed by DAMHAUG [19]. This effort has now been supplemented with sparse versions of the eigensolvers MSSIT and LANCZ2, called SPRSIT and SPRLAN, respectively.

Apart from the storage format, the sparse based subroutines are identical to the corresponding skyline based subroutines, and for all operations that do not involve the system matrices explicitly they use the same subroutines. A fair number of problems have been analyzed with both versions of both algorithms, and for all practical purposes they give identical results.

The purpose of this section is to compare the versions with respect to efficiency, in terms of both computer times and storage requirements.

Three frames, A-1, A-6 and C-9, have been analyzed, each for two values of m=q, namely 5 and 100. We have used ‘standard’ values for the various parameters. For subspace iteration the number of trial vectors, p, are 15 and 150, respectively, and for truncated Lanczos we have used KPD = 1 (Cholesky factorization of A), KORT = 1 (full, ‘one-pass’ reorthogonalization) and KEX = 2. In all cases pseudo-random start vectors are used, and the relative error tolerance parameter, ε, has been set to $10^{-10}$ throughout.

Measured CPU-times, using Computer IV, are summarized in Table 7.11, and the total amount of storage used, measured in terms of words, is shown in Table 7.12. We see that, apart from the smallest problem (Frame A-1), the sparse storage format is more efficient, in terms of both time and storage. The numbers seem to indicate much the same tendency for both algorithms. For the smallest problem there is very little in it; the sparse storage scheme is slightly less efficient. For the largest problem, however, the sparse storage format cuts the CPU-time by about 20%. This is of course quite significant, but it is not nearly as large a reduction as has been observed for straightforward solution of equations, for similar problems. This is perhaps not surprising since some of the operations are inde-
Table 7.11: CPU-times, as function of storage format, for some unsymmetric frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>Type</th>
<th>$n$</th>
<th>$m=q$</th>
<th>CPU-time (in seconds) - Computer IV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Subspace iteration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Skyline (MSSIT)</td>
</tr>
<tr>
<td>A-1</td>
<td>240</td>
<td>5</td>
<td></td>
<td>0,430</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>55,8</td>
</tr>
<tr>
<td>A-6</td>
<td>2940</td>
<td>5</td>
<td></td>
<td>39,8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>1370</td>
</tr>
<tr>
<td>C-9</td>
<td>6006</td>
<td>5</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>9700</td>
</tr>
</tbody>
</table>

Table 7.12: Total storage requirements, as function of storage format, for some unsymmetric frames

<table>
<thead>
<tr>
<th>Frame</th>
<th>Type</th>
<th>$n$</th>
<th>$m=q$</th>
<th>Total storage requirement (in words)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Subspace iteration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Skyline (MSSIT)</td>
</tr>
<tr>
<td>A-1</td>
<td>240</td>
<td>5</td>
<td></td>
<td>44,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>308,180</td>
</tr>
<tr>
<td>A-6</td>
<td>2940</td>
<td>5</td>
<td></td>
<td>2,699,630</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>4,421,690</td>
</tr>
<tr>
<td>C-9</td>
<td>6006</td>
<td>5</td>
<td></td>
<td>8,873,342</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td></td>
<td>12,251,042</td>
</tr>
</tbody>
</table>

The total CPU-time also depends very much on matrix vector multiplication. Comparing skyline and sparse storage formats, with respect to solution of typical finite element equilibrium equations, shows that the larger the problem the more superior becomes the sparse storage format. We see the same tendency here, although to a much smaller degree.

For one of the problems, Frame A-6 with $m = q = 100$, an interesting observation was made for the subspace iteration subroutines. The skyline based version, MSSIT, found the 100 eigenvalues and
corresponding eigenvectors in 24 iterations, whereas the sparse version, SPRSIT, for some reason or other needed 26 iterations to find the required number of eigenpairs; it found in fact 101 acceptable pairs. Everything, except the storage format, is the same, and for all practical purposes the two subroutines found the same eigenvalues (and eigenvectors). All matrices, including the pseudo-random start vectors, are identical, within machine precision. However, the order in which the numbers are operated upon is different for some of the numerical operations. This seems to be the only explanation for the rather surprising difference in behavior, and as such shows that the robust procedure is also sensitive to the sequence of operations. In spite of the difference in iteration cycles, the sparse based subroutine is noticeably faster.

We see roughly the same trends for storage requirements as for solution times, and it seems fair to state that the sparse storage format is superior for all problems of a size for which efficiency matters.

7.6 Conclusions and recommendations

When summarizing the findings in this chapter it should be kept in mind that all computations reported in the preceding sections are related to the same type of problem: free, undamped vibrations of 3D skeletal frames. We do have some limited experience with linearized buckling problems of frame type structures that seems to support the results of this report. Nevertheless, it is perhaps wise to avoid sweeping statements and categoric conclusions.

All results reported here suggest that both the subspace iteration and the truncated Lanczos algorithms, as implemented here, are reliable, accurate and robust general purpose eigensolvers for the type of problems encountered in structural engineering.

*If speed of computation is important, there can be no doubt that the truncated Lanczos algorithm is superior to the subspace iteration method.*

This is true if only a few eigenvalues/eigenvectors are requested, in which case we see a factor of not less than 3 between the two methods, and it becomes more evident the more eigenpairs are requested. For 100 and more eigenpairs a factor of 20, in favor of Lanczos, is not unusual.

It should be emphasized that the implementations presented here do not claim to be the fastest possible. While speed is important, and both MSSIT (SPRSIT) and LANCZ2 (SPRLAN) are believed to be reasonably efficient implementations, reliability, applicability to different types of problems (general purpose) and ease of use are
also important design criteria. Without offering any proof, a survey of the literature seems to indicate that the truncated Lanczos algorithm has as much, if not more, potential for speed up than has subspace iteration.

The Lanczos algorithm also has the edge in terms of storage requirements, although here the advantage is not nearly as dramatic. In terms of the basic matrix storage format, skyline or sparse, it has been demonstrated that the sparse format is superior to skyline for all problems of a certain size, and more so the larger the matrices are. This applies to both computer time and storage. However, this advantage of the sparse format is not nearly as significant for solution of eigenvalue problems as it is for straightforward solution of systems of linear equations.

The remainder of this section deals with recommendations for the use of the individual subroutines.

### Subspace iteration - MSSIT / SPRSIT

The user has to indicate which of the two matrices, $A$ or $B$, is most likely to be positive definite. This information is used by the subroutine to determine which reduced (subspace) eigenproblem to solve, the problem defined by Eq. (3.8) or the inverse problem (one of the reduced matrices, $\bar{A}_{k+1}$ or $\bar{B}_{k+1}$, must be positive definite in order for GENHQL to provide a solution). In most cases the $A$ matrix will be the stiffness matrix, and (in linear theory) this matrix is positive definite. There is, however, one situation that may render $A$ indefinite, and that is a free vibration problem with a prescribed positive shift value. For such a problem it is recommended to point at $B$ as the positive definite matrix - a consistent mass matrix is positive definite, and so is a diagonalized mass matrix, whereas a lumped mass matrix is semi-definite and will not guarantee that $\bar{B}_{k+1}$ is positive definite.

As for the initial values of the trial vectors, that is the start vectors, pseudo-random vectors are strongly recommended.

Unless storage is a limiting factor, a somewhat higher number of trial vectors ($p = NIV$) than that often quoted in the literature is recommended. This is clearly a problem dependent issue, but in our experience a value for $p$ defined as

$$p = m + \text{int}\{m^{68}\} + 3$$

where $m$ is the number of eigenvalues requested, seems to be reasonable in most cases.

The available modified, and faster, procedure, which takes advantage of already converged eigenvalues/eigenvectors, is not recom-
**NUMERICAL RESULTS**

mended. In order for this procedure to produce good accuracy, a fairly small value need to be specified for the relative error tolerance parameter $\varepsilon$. Both overall accuracy and speed is better served by the standard procedure in combination with a larger value for $\varepsilon$. Recommended values for $\varepsilon$ are in the range $10^{-6}$ to $10^{-8}$. The higher value of $10^{-6}$ is probably sufficient for most practical problems, particularly if more than 10 eigenvalues/eigenvectors are requested, since the lower eigenvalues and corresponding eigenvectors will then be determined with significant less relative error.

It should be emphasized that the default value for the maximum number of iterations (MAXIT), currently set to 20, may be too low, particularly if a fairly large number of eigenvalues are requested. In such a case it is, for a structural problem, likely that the eigenvalues at the high end of the requested cluster are closely spaced, thus forcing many iteration cycles. In some such cases more than 50 iterations have been necessary to secure a complete solution.

**Truncated Lanczos - LANCZ2 / SPRLAN**

The user has to specify which matrix to Cholesky factorize, A or B. The choice should always be A provided it is positive definite. Since A will usually be the stiffness matrix this requirement is satisfied in all linear problems except in free vibration problems for which a positive shift value is specified, in such cases the resulting $A_s$ matrix, see Eq. (3.43), may well be indefinite. It should be noted that in this case factorization of B (into $U_s^T U_s$) will always work, also for a lumped mass matrix (in which case $U_B^T = B^{1/2}$).

A pseudo-random start vector is recommended in all cases. As for reorthogonalization, SIMON's partial (KORT = -1) or the simple 'half' (KORT = -2) reorthogonalization procedure is recommended for most cases. Since these more 'economic' procedures have not been tested for very large values of m (= NEVAL), that is in excess of 100, perhaps a full, 'single-pass' reorthogonalization (KORT = 1) should be used for such problems.

The most difficult parameter to specify is the $s (= KEX)$ parameter associated with the stop criteria, see page 54. This parameter must have a value between 1 and 5. The maximum value is set arbitrarily. The worst that can happen by specifying a large value for $s$ is that the process probably uses more computational effort (more Lanczos steps) than necessary (a certain amount of 'over-kill'). For 'normal' problems a value of 2 seems to work quite well; a lower value (that is 1) is not recommended. If there is a real chance of multiple eigenvalues in the problem (normally associated with symmetry), a higher value of 3 or 4, or even 5 may be considered. How-
ever, as demonstrated by one particular example, even \( KEX = 5 \) may not guarantee that the procedure will find the largest multiple eigenvalue in the requested cluster. If it is absolutely vital that the subroutine finds all requested eigenvalues within the specified range (defined by a given number, \( m \)), it is recommended to use a value of 3 or 4 for \( KEX \), and at the same time ask for a few eigenvalues more than the required number, that is increase \( m \) by 2 to 4 (thus forcing more Lanczos steps to be carried out). This may sound more alarming than it really is. It is very unlikely that more than one multiple eigenvalue will be missed (if any - it has only happened once!), and it will be one of the highest eigenvalues in the cluster. Also, all eigenvalues returned by the subroutine are most definitely true eigenvalues (and the corresponding vectors, if requested, are true eigenvectors), but the last values may reside slightly higher in the spectrum than suggested by the subroutine.

The maximum number of Lanczos steps, \( p = \text{MAXLAN} \) should be in the range of 2 times \( m = \text{NEVAL} \) for large values of \( m \). However, for small values of \( m \) it must be greater. For \( m = 1 \), for instance, a value of 10 or more may be necessary for \( p \). A makeshift formula for \( p \), based on the findings in this work, may look something like

\[
p = \text{int} \{ m + 10\sqrt{m} \}
\]


Index

A
acceptable eigenvalues (Lanczos) 54
accuracy 64

B
backsubstitution 35
Bathe, K.-J. 18, 40, 41, 106, 107
buckling factor 1

C
change of problem
   inversion 22
   shift 21
characteristic polynomial 6
Cholesky factorization 8
computational efficiency 62
computer
   hardware 68
   software 68
CPU-time 62

D
Damhaug, A.C. 131
determinant methods 28
double precision 130

efficiency 62
EIG 61
eigenpair 6
eigenproblem
   generalized 5
   special (standard) 6
eigenvalue 1, 6
eigenvalue characterization
   minimax 12
   recursive 12
eigenvector 1, 6
   orthogonality 9
EISPACK 72
error tolerance 39, 49, 99, 114

F
Falk, S. 31, 84
Felippa, C.A. 73
forward substitution 35
frame shapes 66
frame types 66
Francis, J.G.F. 32
free vibration 1, 64

G
good eigenvalues (Lanczos) 54
Gram-Schmidt orthogonalization 15
   modified procedure 17
   standard procedure 16

good eigenvalues (Lanczos) 54
Givens tridiagonalization 31
Golub, G.H. 8
Golub, G.H. 8

H
Householder tridiagonalization 31

J
Jennings, A. 8, 18

K
KEX 95
KORT 118
KPD 128
Krylov
   matrix 19
   sequence 19
LANCZ2 94
Lanczos method 44
accuracy and robustness 111
basic algorithm 45
breakdown 48
computational aspects 57
computational effort 123
efficiency 127
eigenvalue extraction 53
finding the eigenvectors 58
generalized problem 54
reorthogonalization 50, 118
start vector 53
stop criteria 54
storage requirement 130
with shift 55
Lanczos step 54
Lanczos vectors 44
Lanczos, C. 44
Langemeyer, P. 31, 84
linearized buckling 1

M
machine precision 52
mass matrix 1
mass model
consistent 2
diagonalized 2
lumped 2
matrix
diagonal 7
diagonal dominant 7
Hessenberg 7
identity 6
indefinite 7
inverse 10
lower triangular 7
orthogonal 7
positive definite 7
positive semi-definite 7
singular 7
skew-symmetric 7
symmetric 7
trace of 8
transpose 7
tridiagonal 7
unit 6
unit triangular 7
upper triangular 7
matrix condition number 80
matrix factorization
Cholesky 7
L D L-transpose 7
orthogonal 17
spectral (decomposition) 9
MAXIT 93
MAXLAN 94
MSSIT 93

N
NEQ 66
NEVAL 93
NEVEC 94
Newman, M. 45
NIV 93

O
Ojalvo, I.U. 45

P
Paige, C.C. 45, 47, 50, 52
Parlett, B.N. 8, 31, 52
polynomial iteration methods
determinant search 28
sign count 28
positive definite 7

Q
QR transformation 32

R
Rayleigh
principle 13
quotient 11
theorem 12
Rayleigh-Ritz 13
renumbering 68
reorthogonalization
full 50
half 52
partial 50
Ritz
basis vectors 13
coordinates 13
values 14
vectors 14

S
SAM library 2, 61
Sehmi, N.S. 37, 44, 45, 52
shift value 21
similarity transformation 10
Simon, H.D. 52, 120, 135
stiffness matrix 1
storage arrangement 131
storage format
   skyline 62
   sparse 62
structural dynamics 1
Sturm sequence check 18

subroutine
   BSCHOL 84
   FACHOL 84
   FSCHOL 84
   GENHQL 84
   GENJAC 84
   IOU3D 72
   HQLS 72
   HQRIS 72
   IIS3D 72
   IMTQL1 72
   IMTQI2 72
   LANCZ2 94
   MSSIT 93
   QLS3D 72
   QLS3DV 72
   QRS3D 72
   SPRLAN 94
   SPRSIT 93
   SPSSIJC 73
   SSSIT 92

subspace iteration 37
   accuracy and robustness 96
   basic algorithm 37
   computational effort 100
   computational procedure 41
   convergence 39
   efficiency 127
   number of trial vectors 40, 106
   reduced eigenproblem 43
   speed up 44, 108
   start vectors 40, 107

storage requirement 130
   with shift 39
subspace matrices 38

T
test frame 64
time measurements 63
time ratio 127
transformation
   orthogonal 10
   similarity 10
   to standard form 22
transformation methods 29
   generalized Jacobi 31
   Givens tridiagonalization 31
   Householder tridiagonalization 31
   Jacobi diagonalization 30
   QR and QL methods 32
   Rutishauser’s L.R method 32
truncated Lanczos 44

V
Van Loan, C.F. 8
vector iteration methods 34
   inverse iteration 35
   power iteration 34
   subspace iteration 37
vibration
   free, undamped 1
   frequency 1
   mode shape 1
Vissing, S. 31, 90

W
Wilkinson, J.H. 27, 33
Wilson, E.L. 18, 40, 41, 106, 107