THE NUMERICAL DETERMINATION OF THE EIGENVALUES AND EIGENVECTORS OF LARGE ORDER SPARSE MATRICES

BY

CHRISTOPHER COLLARD RICK, B.Sc.

A Doctoral Thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of the Loughborough University of Technology

Supervisor: PROFESSOR D.J. EVANS
DEPARTMENT OF COMPUTER STUDIES

DECLARATION

I declare that the following thesis is a record of research work carried out by me, and that the thesis is of my own composition. I also certify that neither this thesis nor the original work contained herein has been submitted to this or any other institution for a degree.

C.C. RICK.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Evans for giving me the opportunity, his guidance and ideas, and a good push when I needed it.

I am grateful to Drs. Dunbar, Blakemore and Barlow for many useful and productive discussions. More intangible but equally important, was the help of H.J. Gould and my parents. Finally I would like to thank Miss J.M. Briers whose typing was considerably more than expert.

CONTENTS

PAGE

CHAPTER 1

INTRODUCTION

1.1

The aim of the ensuing work is to present a number of numerical methods which can be used to solve the algebraic eigenproblem,

$$A\underline{x} = \lambda\underline{x} \quad , \tag{1.1.1}$$

where A is a known N×N matrix. The scalar λ is an eigenvalue of the matrix A, and the N-vector $\underline{x}$ is the corresponding eigenvector.

Equation (1.1.1) is the matrix notation for a set of N linear equations in N unknowns x_i, i=1,2,...,N,

$$\left.\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots\ldots + a_{1,N}x_N &= \lambda x_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots\ldots + a_{2,N}x_N &= \lambda x_2 \\ &\vdots \\ a_{N,1}x_1 + a_{N,2}x_2 + \ldots\ldots + a_{N,N}x_N &= \lambda x_N \end{aligned}\right\} \tag{1.1.2}$$

The equations are homogeneous and therefore a trivial solution $\underline{x}=0$ exists. However, for certain critical values of the parameter, the equations(1.1.2) have a finite non-trivial solution in which the relative values of the variables are defined, but not their absolute values. The parameter is λ, the eigenvalue, and the variables make up the eigenvector $\underline{x}$. These values are important, as if for example equations (1.1.2) represent some physical system then the eigenvalues and eigenvectors are significant features of the system.

Many problems arise in engineering and physics that are defined by partial differential equations. It is the numerical solution of the partial differential equations by finite difference methods that usually present an equation of the form (1.1.1) to be solved. Eigenvalue problems arise less frequently from other sources such as economics, and information system design.

1.2 SOME EXAMPLES OF EIGENVALUE PROBLEMS

The first physical problem to be considered is that of the small vibrations of particles on a string under tension. To facilitate presentation of the idea simplifications are made so that the string is assumed weightless and uniform, with no gravity acting, and motion perpendicular to the rest position of the string. Attached to the string are five unequal weights equally spaced, and the string is under tension P as shown in Figure (1.1.1)

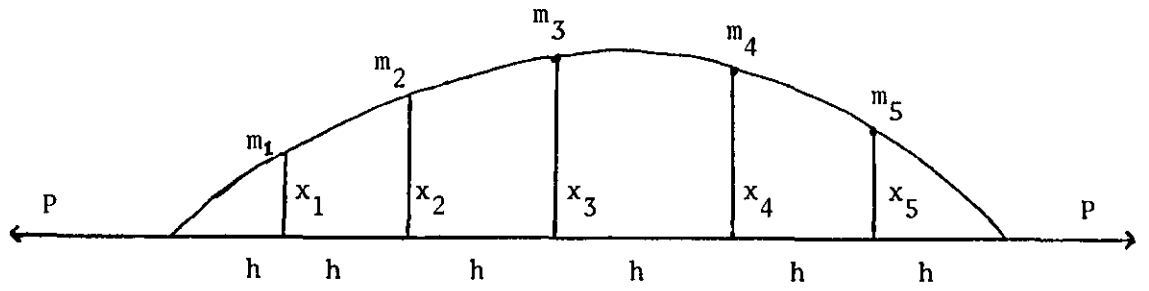


FIGURE 1.1.1

Applying Newtons 3rd law of motion to each mass in turn yields the following equations,

$$\left.\begin{aligned}
m_1 \frac{d^2x_1}{dt^2} &= \frac{-Px_1}{h} + P\frac{(x_2-x_1)}{h} \\
m_2 \frac{d^2x_2}{dt^2} &= \frac{-P(x_2-x_1)}{h} + P\frac{(x_3-x_2)}{h} \\
m_3 \frac{d^2x_3}{dt^2} &= \frac{-P(x_3-x_2)}{h} - P\frac{(x_3-x_4)}{h} \\
m_4 \frac{d^2x_4}{dt^2} &= +P\frac{(x_3-x_4)}{h} - P\frac{(x_4-x_5)}{h} \\
m_5 \frac{d^2x_5}{dt^2} &= +P\frac{(x_4-x_5)}{h} - P\frac{x_5}{h}
\end{aligned}\right\} \qquad (1.2.1)$$

Letting

$$\underline{x}^T = (x_1, x_2, x_3, x_4, x_5) \quad , \tag{1.2.2}$$

and,

$$d_i = m_i h/P, \qquad i=1,2,\ldots,5 \ , \tag{1.2.3}$$

the system can be written in matrix notation as

$$D \ \frac{d^2\underline{x}}{dt^2} = T \ \underline{x} \quad , \tag{1.2.4}$$

where,

$$D = \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \ , \tag{1.2.5}$$

and,

$$T = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \ . \tag{1.2.6}$$

If the system vibrates with all masses moving in phase or in direct opposition, i.e. normal mode operation then,

$$\frac{d^2\underline{x}}{dt^2} = -w^2\underline{x} \quad . \tag{1.2.7}$$

Substituting equation (1.2.7) in (1.2.4) gives,

$$D \ w^2 \ \underline{x} = -T\underline{x} \quad , \tag{1.2.8}$$

where the eigenvalues w_1, w_2, w_3, w_4, w_5 are the normal frequencies of the motion. Equation (1.2.7) is not exactly of the same form as (1.1.1) but can easily be transformed to that form by methods described in Chapter 2.

The next problem to be considered is that of an axially loaded beam that can rotate at both ends, with the only unconstrained deflection being vertically at the top, as in Figure (1.2.1), with load P,

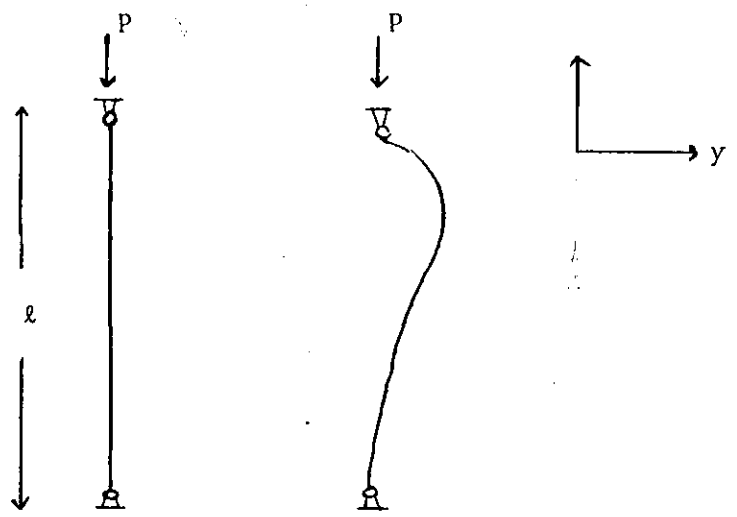


FIGURE 1.2.1

Assuming the column, of length ℓ, is in equilibrium with small lateral deflections $y(x)$ and has a bending stiffness EI, it can be shown that the following differential equation and end conditions must be obeyed,

$$\left.\begin{aligned} &\frac{d^2y}{dx^2} + \frac{Py}{EI} = 0 \\ &\text{and} \\ &y = 0 \text{ at } x = 0 \\ &y = 0 \text{ at } x = \ell \end{aligned}\right\} \qquad (1.2.9)$$

To achieve a numerical solution to equation (1.2.9) a finite difference approximation to the continuous system can be used. The column is divided into segments and the displacements (y_i, i=1,....,5) are considered at a number of discrete points, as in Figure (1.2.2)

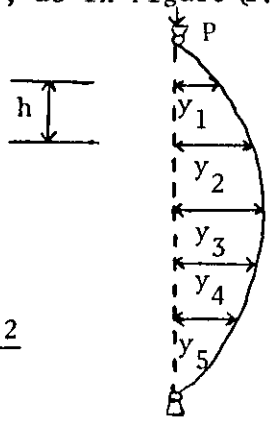


FIGURE 1.2.2

The Taylor series expansion for displacement (y(x)) in terms of values of the function at adjacent points is,

$$y(x+h) = y(x) + \frac{h dy(x)}{dx} + \frac{h^2}{2} \frac{d^2y(x)}{dx^2} + \ldots\ldots, \qquad (1.2.10)$$

$$y(x-h) = y(x) - \frac{h dy(x)}{dx} + \frac{h^2}{2} \frac{d^2y(x)}{dx^2} + \ldots\ldots, \qquad (1.2.11)$$

Adding equations (1.2.10) and (1.2.11) gives,

$$y(x+h)+y(x-h) = 2y(x) + h^2 \frac{d^2y(x)}{dx^2} + O(h^4) \quad . \qquad (1.2.12)$$

The error term of order (h^4) in (1.2.12) is due to truncation of the series after four terms in the previous two equations. Re-arranging equation (1.2.12) and ignoring the error term yields the finite difference approximation,

$$\frac{1}{h^2} (y(x+h)-2y(x)+y(x-h)) = \frac{d^2y(x)}{dx^2} \quad , \qquad (1.2.13)$$

which can now be written as,

$$\frac{1}{h^2}(y_{i+1}-2y_i+y_{i-1}) = \frac{d^2y_i}{dx^2} \quad , \; i=1,2,\ldots 5 \; . \qquad (1.2.14)$$

Equation (1.2.13) can now be substituted into equation (1.2.9) to yield,

$$(-y_{i+1}+2y_i-y_{i-1}) = \frac{Ph^2}{EI} y_i, \quad i=1,2,\ldots,5 \; , \qquad (1.2.15)$$

The end or boundary conditions defined by equation (1.2.9) indicate that $y_0=y_6=0$, while the right hand side of (1.2.15) can be simplified to,

$$\frac{Ph^2}{EI} y_i = \lambda y_i, \qquad i=1,2,\ldots,5. \qquad (1.2.16)$$

The set of difference equations (1.2.13) can now be written in matrix form as,

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \lambda \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \quad . \qquad (1.2.17)$$

This can be written in matrix notation as equation (1.1.1), where the eigenvalues represent the buckling loads.

With this type of problem N can become very large by making h smaller or having larger, giving rise to large sparse matrices. An analytical solution to equation (1.2.17) is already known, but configurations for which classical solutions do not exist can be produced by having beams of non-uniform stiffness, or axial loads varying with x.

For systems that can be defined by a partial differential equation over a given domain a finite difference approximation to the equations at grid points on this domain will result in a set of linear simultaneous equations to be solved. For equilibrium equations or steady state problems such as those defined, for example, by Laplace's equation applied (say) to the steady flow of incompressible non-viscous fluid,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial U^2}{\partial y^2} = 0 \ , \qquad (1.2.18)$$

and the heat conduction equation for (say), heat flow in a bar or rod,

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} \ , \qquad (1.2.19)$$

an eigenvalue problem does not arise.

Eigenvalue problems can be considered as extensions of equilibrium problems in which critical values of certain parameters are required in addition to the steady state configuration. Most examples of this kind of problem can be found in problems of buckling and stability of structures, resonance in acoustics and electrical circuits, and natural frequency problems in vibrations of systems. A finite difference method of solution will then result in a set of equations of the same form as equation (1.1.1).

CHAPTER 2

BASIC LINEAR ALGEBRAIC THEORY

2.1 INTRODUCTION

In this chapter is presented some of the results of basic matrix eigenvalue and eigenvector theorems. No attempt is made to give rigorous proofs as they are well known and can be found in the literature. References are given where needed, but for a comprehensive treatment of the relevant subject matter the recommended reading is Wilkinson (1965) and Hohn (1964).

Some of the methods that are modified and used to obtain eigenvalues in later chapters are described here in their original form. Also, some of the current methods being used to determine the eigenvalues of dense matrices and sparse band matrices are also outlined. Their advantages and disadvantages in dealing with sparse matrices are briefly described, and it is these methods as programmed in the N.A.G. library that are used to compare with results obtained from the new algorithms outlined in later chapters.

2.2 NOTATION

A matrix will always be denoted by a capital letter, most commonly used are the letters C and A. In general where possible C is used to denote, real symmetric matrices and A for a general matrix which may be symmetric or unsymmetric and/or complex. All of the matrices in the work are square, and unless specifically noted have N rows and N columns (N×N matrices). Unless the elements of the matrix are explicitly defined the matrix elements, for matrix A say, are denoted by $a_{i,j}$ where i denotes position in the i^{th} row and position in j^{th} the j^{th} column.

The determinant of a matrix will be written either as det (A) or $|A|$, and the matrix is said to be singular if the determinant equals zero. This can be written,

$$\det (A) = |A| = 0 \qquad (2.2.1)$$

If the matrix A is non-singular then its inverse (written as A^{-1}) exists and is defined by,

$$A\,A^{-1} = A^{-1}\,A = I \quad . \tag{2.2.2}$$

Matrices with large numbers of zero elements are cumbersome and difficult to write and collections of large zero elements are replaced by a space with a large 0 in it. So that if matrix C is a 5×5 symmetric tridiagonal matrix thus,

$$C = \begin{bmatrix} c_1 & b_2 & 0 & 0 & 0 \\ b_2 & c_2 & b_3 & 0 & 0 \\ 0 & b_3 & c_3 & b_4 & 0 \\ 0 & 0 & b_4 & c_4 & b_5 \\ 0 & 0 & 0 & b_5 & c_5 \end{bmatrix} , \tag{2.2.3}$$

it would be written for simplicity as,

$$C = \begin{bmatrix} c_1 & b_2 & & & \\ b_2 & c_2 & b_3 & 0 & \\ & b_3 & c_3 & b_4 & \\ & 0 & b_4 & c_4 & b_5 \\ & & & b_5 & c_5 \end{bmatrix} . \tag{2.2.4}$$

Likewise if A is a 6×6 upper triangular matrix thus,

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ 0 & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ 0 & 0 & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ 0 & 0 & 0 & a_{4,4} & a_{4,5} & a_{4,6} \\ 0 & 0 & 0 & 0 & a_{5,5} & a_{5,6} \\ 0 & 0 & 0 & 0 & 0 & a_{6,6} \end{bmatrix} , \tag{2.2.5}$$

it would be written for simplicity as,

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ & & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ & & & a_{4,4} & a_{4,5} & a_{4,6} \\ & 0 & & & a_{5,5} & a_{5,6} \\ & & & & & a_{6,6} \end{bmatrix} . \qquad (2.2.6)$$

Vectors are represented as underlined lower case letters if they are column vectors. Elements of that vector have the same letter, but a lower suffix giving the position of that element in the vector. The corresponding row vector has an upper suffix T. For example if $\underline{x}$ is the vector,

$$\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} , \qquad (2.2.7)$$

then x^T is the vector,

$$\underline{x}^T = (x_1, x_2, x_3) . \qquad (2.2.8)$$

Eigenvalues are denoted by lower case Greek letters, and the one most commonly used is λ (lambda). If I is the (N×N) identity matrix,

$$I = \begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ 0 & & & 1 \end{bmatrix} , \qquad (2.2.9)$$

then the matrix λI is written,

$$\lambda I = \begin{bmatrix} \lambda & & & 0 \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \\ 0 & & & \lambda \end{bmatrix} . \qquad (2.2.10)$$

The matrix λI is a diagonal matrix with the only non-zero elements on the main diagonal.

The transpose of a matrix A (say) is denoted by A^T, and is the matrix whose element in the i^{th} row, j^{th} column is $a_{j,i}$. If the matrix A has complex elements and $a^*_{i,j}$ is the complex conjugate of $a_{i,j}$ then the matrix A* (the conjugate transpose) is the matrix whose element in the i^{th} row, j^{th} column is $a^*_{j,i}$.

From the above a number of special matrices can be defined. For a Hermitian matrix A,

$$A^* = A \quad . \tag{2.2.11}$$

If a matrix C is symmetric,

$$C^T = C \quad . \tag{2.2.12}$$

If a matrix A is unitary,

$$A^*A = I \quad . \tag{2.2.13}$$

Finally a real orthogonal matrix A is such that,

$$A^TA = I \quad . \tag{2.2.14}$$

2.3 EIGENVALUES

The basic algebraic eigenproblem is the determination of λ such that,

$$A\underline{x} = \lambda\underline{x} \quad , \tag{2.3.1}$$

has a non-trivial solution, where A is a N×N matrix and $\underline{x}$ is an N vector then this can be written as,

$$(A-\lambda I)\underline{x} = 0 \quad . \tag{2.3.2}$$

It can be shown that there is a non-trivial solution to this problem if and only if the determinantal equation,

$$\det (A-\lambda I) = 0 \tag{2.3.3}$$

is satisfied. The determinant of equation (2.3.3) can be expanded by the Laplace expansion to yield,

$$\alpha_0 + \alpha_1\lambda + \alpha_2\lambda^2 + \ldots + \alpha_{N-1}\lambda^{N-1} + (-1)^N\lambda^N = 0 \ . \qquad (2.3.4)$$

This is called the characteristic equation of the matrix A. The N roots of the polynomial (2.3.4) are the N eigenvalues of the matrix A. Corresponding to each eigenvalue of the matrix A, (λ), there is at least one non-trivial vector $\underline{x}$ for which (2.3.1) is satisfied. This is called an eigenvector of A corresponding to λ.

The elements of A may be complex, but if the matrix A is Hermitian then all the eigenvalues will be real. If the matrix A is unsymmetric then some or all of the eigenvalues may be complex.

There are methods for determining eigenvalues, some of which are described later, which work with the matrix equation (2.3.1) or directly on the matrix A whereas the methods developed in later chapters are all concerned with finding the roots of the characteristic equation (2.3.4). This presents a different emphasis in approach, where, provided the characteristic equation can readily be found the structure of the matrix is no longer important and thus economies in space can be made inside the computer.

Similarity Transformations

If a matrix A is transformed to $R^{-1}AR$ where R is a non-singular matrix (i.e., the inverse of R exists) then this is known as a similarity transformation. The matrices A and $R^{-1}AR$ are said to be similar. Of particular importance (as discussed in section 5) is the transformation when R is a unitary matrix. In this case the matrices A and $R^{-1}AR$ are said to be unitarily similar.

The usefulness of such a transformation is that the eigenvalues of a matrix are invariant under the transformation. Therefore, similar matrices have the same eigenvalues. This can easily be shown for if

$$A\underline{x} = \lambda\underline{x} \ , \qquad (2.3.5)$$

then,

$$R^{-1}A\underline{x} = \lambda R^{-1}\underline{x} \quad , \tag{2.3.6}$$

and,

$$(R^{-1}AR)R^{-1}\underline{x} = \lambda R^{-1}\underline{x} \quad . \tag{2.3.7}$$

Clearly the eigenvalues are unchanged and the eigenvectors are pre-multiplied by R^{-1}.

It can easily be shown that any N×N matrix with N linearly independent eigenvectors is similar to a diagonal matrix with the N eigenvalues on the main diagonal. The more general result from Schür's theorem is that any square matrix is unitary similar to a triangular matrix with the eigenvalues on the diagonal. So if a unitary matrix or a sequence of unitary matrices can be found that transform a matrix to triangular form then the eigenvalues appear on the main diagonal. It is this kind of transformation that is the basis of many methods for determining eigenvalues.

The Jordan Canonical Form

As a general matrix cannot always be reduced to diagonal form it is of interest to know what is the most compact form a matrix can be reduced to by similarity transformations. The most compact form for a general matrix is the Jordan canonical form which, while of little practical importance, does help to illustrate the system of eigenvalues and eigenvectors of a matrix.

First a sequence of matrices are defined,

$$J_1(\lambda) = [\lambda] \quad , \tag{2.3.8}$$

$$J_r(\lambda) = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & 1 & \\ & & \ddots & \ddots \\ 0 & & & 1 \\ & & & \lambda \end{bmatrix} \quad , \tag{2.3.9}$$

where r>1, and $J_r(\lambda)$ is an (r×r) matrix with an eigenvalue λ of multiplicity r, but only one eigenvector $\underline{x}$ where,

$$\underline{x}^T = (1,0,0,.....,0) \quad . \tag{2.3.10}$$

The matrix $J_r(\lambda)$ is called a simple Jordan submatrix of order r.

If matrix A is a N×N matrix with s distinct eigenvalues $\lambda_1,\lambda_2,.....,\lambda_s$ of multiplicities $m_1,m_2,....,m_s$ where

$$\sum_{i=1}^{s} m_i = N \quad , \tag{2.3.11}$$

the following theorem can be stated.

There exists a non-singular matrix R such that $R^{-1}AR$ has simple Jordan submatrices $J_r(\lambda_i)$ isolated along the diagonal with all other elements equal to zero. If there are p submatrices of orders r_j, j=1,2,....p associated with any λ_i then,

$$\sum_{j=1}^{p} r_j = m_i \quad . \tag{2.3.12}$$

The matrix $R^{-1}AR$ is called the Jordan canonical form of A, and is unique apart from the ordering of the submatrices along the diagonal.

If a (N×N) matrix has N distinct eigenvalues the Jordan submatrices are all of order 1, and the matrix can be reduced to a diagonal matrix. If a matrix has fewer than N distinct eigenvalues, and fewer than N linearly independent eigenvectors the matrix is said to be defective.

A matrix for which there is more than one Jordan submatrix (this implies there is also more than one eigenvector) associated with λ_i for some value of i is said to be derogatory. Matrices that are derogatory and/or defective present particular problems when solving the eigenvalue problem.

Vector and Matrix Norms

It is useful to have some measure of the 'size' of a vector or matrix analogous to the modulus of a complex number. This is achieved

by the use of norms. The norm of a vector is denoted by $||\underline{x}||$ and satisfies the relations,

$$||\underline{x}|| > 0, \text{ unless } \underline{x} = 0 \quad , \tag{2.3.13}$$

$$||k\underline{x}|| = |k| \; ||\underline{x}||, \text{ where k is a complex scalar,} \tag{2.3.14}$$

$$||\underline{x}+\underline{y}|| \leq ||\underline{x}||+||\underline{y}||. \tag{2.3.15}$$

The general form of a vector norm is given by,

$$||\underline{x}||_p = (|x_1|^p+|x_2|^p+.....+|x_N|^p)^{1/p}, \quad (p=1,2,\infty). \tag{2.3.16}$$

The norms that are of most use are for when, p=2 which is commonly known as the Euclidean length of a vector, and when $p=\infty$ which interpreted as the maximum value of $|x_i|$, i=1,2,...,N.

Similarly the norm of a matrix A is denoted by $||A||$ and satisfies the relations,

$$||A|| > , \text{ unless } A=0 \quad , \tag{2.3.17}$$

$$||kA|| = |k| \; ||A||, \text{ where k is a complex scalar,} \tag{2.3.18}$$

$$||A+B|| \leq ||A||+||B||, \tag{2.3.19}$$

$$||AB|| \leq ||A||.||B||. \tag{2.3.20}$$

For any vector norm there can be defined a corresponding subordinate matrix norm,

$$||A|| = \max_{\underline{x}\neq 0} \frac{||A\underline{x}||}{||\underline{x}||} = \max_{||\underline{x}||=1} ||A\underline{x}|| \; . \tag{2.3.21}$$

Therefore, the subordinate matrix norm satisfies,

$$||A\underline{x}|| \leq ||A|| \; ||x||. \tag{2.3.22}$$

The matrix norms subordinate to vector norms with $p=2,\infty$ are,

$$||A||_2 = (\text{maximum eigenvalue of } A^*A)^{\frac{1}{2}}, \tag{2.3.23}$$

$$||A||_\infty = \max_i \sum_{j=1}^{N} |a_{i,j}| \quad . \tag{2.3.24}$$

Normalised Vector

Not to be confused with a vector norm is the normalised vector, a vector multiplied by a scalar to keep the element size down to 'manageable' figures without changing the direction of the vector. There are several ways of achieving this, the method used in the work is described.

If $\underline{x}$ is an N vector with elements x_i, i=1,2,...,N then calculate

$$s = \left(\sum_{i=1}^{N} x_i^2\right)^{\frac{1}{2}} \quad , \tag{2.3.25}$$

and the normalised vector is,

$$\underline{x}^T = \left(\frac{x_1}{s}, \frac{x_2}{s}, \ldots\ldots \frac{x_N}{s}\right) \quad . \tag{2.3.26}$$

This ensures that the modulus of every element of the vector is less than 1, and also,

$$\underline{x}^T\underline{x} = 1 \quad . \tag{2.3.27}$$

2.4 METHODS FOR OBTAINING EIGENVALUES AND EIGENVECTORS

The Power Method and Rayleigh Quotient

If A is a (N×N) matrix with linear elementary divisors whose eigenvalues satisfy,

$$|\lambda_1| = |\lambda_2| = \ldots\ldots = |\lambda_r| > |\lambda_{r+1}| \geq \ldots\ldots \geq |\lambda_N| \quad , (r \geq 1). \tag{2.4.1}$$

The eigenvalues $\lambda_1, \lambda_2, \ldots\ldots \lambda_r$ are the dominant eigenvalues. By assumption there exist N linearly independent eigenvectors $\underline{x}_1, \underline{x}_2, \ldots\ldots \underline{x}_N$, any arbitrary vector $\underline{z}_0$ can be expressed in the form,

$$\underline{z}_0 = \sum_{i=1}^{N} \alpha_i \underline{x}_i \quad , \tag{2.4.2}$$

where α_i are scalars, not all zero. The power method is now defined by the simple iterative scheme

$$\underline{z}_k = A\underline{z}_{k-1}, \qquad k=1,2,3,... \tag{2.4.3}$$

Then using equation (2.4.3) repeatedly and substituting using (2.4.2),

$$\underline{z}_k = A\underline{z}_{k-1} = A^2\underline{z}_{k-2} = A^3\underline{z}_{k-3} = \ldots\ldots A^k\underline{z}_0$$

$$= \sum_{i=1}^{N} \alpha_i \lambda_i^k \underline{x}_i \quad . \tag{2.4.4}$$

Provided that $\alpha_1, \alpha_2, \ldots, \alpha_r$ are not all zero the right hand side of equation (2.4.4) is ultimately dominated by the terms $\sum_{i=1}^{r} \alpha_i \lambda_i^k \underline{x}_i$. In particular if r=1 and $\alpha_1 \neq 0$ then,

$$\underline{z}_k = \lambda_1^k \left(\alpha_1 \underline{x}_1 + \sum_{i=2}^{N} \alpha_i (\lambda_i/\lambda_1)^k \underline{x}_i\right)$$

$$= \lambda_1^k (\alpha_1 \underline{x}_1 + \underline{\varepsilon}_k) \quad , \tag{2.4.5}$$

for k sufficiently large, where $\underline{\varepsilon}_k$ is a vector with very small elements. The vector z_k is an approximation to the un-normalised eigenvector and is accurate if $||\varepsilon_k||$ is sufficiently small. To find the corresponding eigenvalue the i^{th} (say) element of two consecutive z_k are used $((\underline{z}_k)i, (z_{k+1})i)$ where,

$$\underline{z}_{k+1} = \lambda_1^{k+1} (\alpha_1 \underline{x}_1 + \underline{\varepsilon}_{k+1}) \quad . \tag{2.4.6}$$

Then,

$$\frac{(\underline{z}_{k+1})i}{(\underline{z}_k)i} = \lambda_1 \frac{(\alpha_1(\underline{x}_1)i + (\underline{\varepsilon}_{k+1})i)}{(\alpha_1(\underline{x}_1)i + (\underline{\varepsilon}_k)i)}$$

$$\to \lambda_1 \quad \text{as} \quad k \to \infty \, . \tag{2.4.7}$$

The Rayliegh quotient of a matrix A for a non-trivial vector $\underline{x}$ is given by,

$$\frac{\underline{x}^* A \underline{x}}{\underline{x}^* \underline{x}} \quad . \tag{2.4.8}$$

It can then be shown that

$$\lambda_1 = \max_{\underline{x} \neq 0} \frac{\underline{x}^* A \underline{x}}{\underline{x}^* \underline{x}} \quad , \tag{2.4.9}$$

where λ_1 is the eigenvalue of A with the largest modulus. The largest eigenvalue of A can therefore be determined using the Rayleigh quotient and a gradient method to optimise (2.4.9).

The Rayleigh quotient can also be used in conjunction with the

power method to determine the eigenvalue of largest modulus.

If $\underline{x}^k$ is the vector obtained by the power method from equation (2.4.3) then,

$$\frac{(\underline{x}^k)^T A \underline{x}^k}{(\underline{x}^k)^T \underline{x}^k} = \lambda_1 + \varepsilon \quad , \tag{2.4.10}$$

where ε is a small error term. In general the Rayleigh quotient corresponding to $\underline{x}^k$ will generally give a better approximation to λ_1 than the power method.

If λ_i is an eigenvalue of matrix A, and $\underline{y}_i$ the corresponding eigenvector then,

$$\frac{\underline{y}_i^T A \underline{y}_i}{\underline{y}_i^T \underline{y}_i} = \lambda_i \quad . \tag{2.4.11}$$

This method is described by Wilkinson (1965) where if an approximation to λ_i is found ($\bar{\lambda}_i$) then the eigenvector corresponding to $\bar{\lambda}_i$ is determined. The relationship (2.4.11) can be used to refine the approximation to λ_i and the eigenvector $\underline{y}_i$.

Gaussian Elimination

Gaussian elimination provides the basis for much of the ensuing work, and so is described in some detail even though it is a widely known and frequently used method. Again a detailed description of the method can be found in Wilkinson (1965).

Gaussian elimination is generally used to obtain the solution of a set of linear equations of the form,

$$A\underline{x} = \underline{b} \quad , \tag{2.4.12}$$

where A is a dense matrix and $\underline{b}$ a known vector. This is achieved in practice by Gaussian elimination on the matrix A with the vector $\underline{b}$ included as the $(n+1)^{th}$ column. The elimination in A will now be described.

The elimination in the matrix A takes place in N-1 stages,

working on the matrix A^i, i=0,1,...,N-2, with elements $a^i_{j,k}$. After the $(i-1)^{th}$ stage the matrix A^{i-1} has the form,

$$A^{i-1} = \begin{bmatrix} a^{i-1}_{1,1} & a^{i-1}_{1,2} & a^{i-1}_{1,3} & \cdots & & & a^{i-1}_{1,N} \\ 0 & a^{i-1}_{2,2} & a^{i-1}_{2,3} & \cdots & & & a^{i-1}_{2,N} \\ & 0 & \ddots & & & & \\ & & \ddots & 0 & & & \\ & & & & a^{i-1}_{i,i} & a^{i-1}_{i,i+1} \cdots & a^{i-1}_{i,N} \\ & & 0 & & a^{i-1}_{i+1,i} & a^{i-1}_{i+1,i+1} \cdots & a^{i-1}_{i+1,N} \\ & & & & a^{i-1}_{i+2,i} & a^{i-1}_{i+2,i+1} \cdots & a^{i-1}_{i+2,N} \\ & & & & \vdots & & \\ & & & & a^{i-1}_{N,i} & \cdots & a^{i-1}_{N,N} \end{bmatrix} \quad (2.4.13)$$

Now the elements $a^{i-1}_{i+j,i}$, j=1,2,...,N-i are eliminated by calculating,

$$s_{i+j,i} = - a^{i-1}_{i+j,i} / a^{i-1}_{i,i} \quad , \quad (2.4.14)$$

and adding $s_{i+j,i}$ times row i to row j. It should be noted that after the i^{th} state the elements in row i+1 do not alter value again. The problem arises that the pivotal element $(a^{i-1}_{i,i})$ may be zero or close to zero and the division of (2.4.14) cannot be preformed without overflow or serious growth of rounding errors occurring. There are three ways of avoiding this,

1) Partial pivoting. The i^{th} column is inspected and the element $a^{i-1}_{j,i}$, j=i+1,...,N with largest modulus $(a^{i-1}_{k,i})$ is determined and then row k is interchanged with row i.

2) Total pivoting. The whole of the matrix $a^{i-1}_{j,k}$, j,k>i is inspected and the element with largest modulus is exchanged with $a^{i-1}_{i,i}$ by performing row and column interchanges.

3) The third method involves replacing the zero $a^{i-1}_{i,i}$ by a small element ε, and continuing the process. This procedure is not satisfactory for solving linear equations, but if Gaussian

elimination is being used in some iterative method, this is an acceptable procedure (Viz Ch.5).

The matrix A, ignoring possible interchanges, has now been decomposed to the form,

$$A = LU \quad , \tag{2.4.15}$$

where,

$$U = \begin{bmatrix} a^0_{1,1} & a^0_{1,2} & a^0_{1,3} & \cdots & a^0_{1,N} \\ 0 & a^1_{2,2} & a^1_{2,3} & \cdots & a^1_{2,N} \\ & 0 & a^2_{3,3} & a^2_{3,4} \cdots & a^2_{3,N} \\ & & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \cdots 0 & a^{N-1}_{N,N} \end{bmatrix} \tag{2.4.16}$$

$$L = \begin{bmatrix} 1 & 0 & & & \\ s_{2,1} & 1 & 0 & & \\ s_{3,1} & s_{3,2} & 1 & \ddots & \\ s_{4,1} & s_{4,2} & s_{4,3} & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ s_{N,1} & s_{N,2} & s_{N,3} \cdots & s_{N,N-1} & 1 \end{bmatrix} \tag{2.4.17}$$

Performing Gaussian elimination on matrix A and vector $\underline{b}$ as the $(N+1)^{th}$ column of A from equation (2.4.12) gives,

$$LU\underline{x} = \underline{b} \quad , \tag{2.4.18}$$

$$U\underline{x} = L^{-1}\underline{b} = \underline{d} \quad , \tag{2.4.19}$$

and is known as the forward substitution stage. Now in order to obtain the value of $\underline{x}$ a backward substitution procedure must be performed, and this is best described algorithmically as,

$$\left.\begin{aligned} & x_N = d_N / a_{N,N}^{N-1} \quad , \\ & \text{for } i = N-1, N-2, \ldots\ldots, 1, \text{ do} \\ & x_i = (d_i - \sum_{j=i+1}^{N} x_j a_{i,j}^{i-1}) / a_{i,i}^{i-1} \end{aligned}\right\} \tag{2.4.20}$$

The quantity,

$$V = \prod_{i=1}^{N} a_{i,i}^{i-1} \, (-1)^W \quad , \tag{2.4.21}$$

where W is the number of row and/or column interchanges performed represents the determinant of the matrix A.

Inverse Iteration

Inverse iteration is basically a variation of the power method of the form defined by

$$\left.\begin{aligned} A\underline{y}^{i+1} &= \underline{z}^i \\ \underline{z}^{i+1} &= \underline{y}^{i+1} / ||\underline{y}^{i+1}||_2 \end{aligned}\right\}. \tag{2.4.22}$$

This is the power method using the inverse of A (A^{-1}) and converges to the eigenvector of A corresponding to the eigenvalue of A with smallest modulus. If instead the iteration of (2.4.22) is performed with the matrix $(A-qI)^{-1}$,

$$\left.\begin{aligned} (A-qI)\underline{y}^{i+1} &= \underline{z}^i \quad , \\ \underline{z}^{i+1} &= \underline{y}^{i+1} / ||\underline{y}^{i+1}||_2 \end{aligned}\right\}, \tag{2.4.23}$$

the vector $\underline{z}$ converges to the eigenvector corresponding to the eigenvalue closest to q in the complex plane. Inverse iteration can provide rapid convergence to an eigenvector even if the approximation to the eigenvalue is not good initially.

The implementation of the method, particularly for use on a computer, has been developed by Wilkinson (1965). It can be seen that the iteration (2.4.23) requires the repeated solution of a set of linear equations. This can be achieved using Gaussian elimination with pivoting strategy, Also the LU decomposition of (A-qI) need only

be calculated once making great savings in time during the process. Therefore, ignoring interchanges for clarity of presentation the process becomes,

$$\left.\begin{aligned} L\underline{v} &= \underline{z}^{i} \quad , \\ U\underline{y}^{i+1} &= \underline{v} \quad , \end{aligned}\right\} \tag{2.4.24}$$

where,

$$LU = A-qI \ . \tag{2.4.25}$$

The first step is omitted for i=0 and the second step replaced by

$$U\underline{y}^{1} = \underline{e} \tag{2.4.26}$$

where $\underline{e}$ is the vector whose elements are all 1. This is equivalent to letting $\underline{y}^{0} = L\underline{e}$. This starting procedure works well under most conditions. In general if q is a close approximation to an eigenvalue convergence to the appropriate eigenvector occurs in one or two iterations, then the Rayleigh quotient is determined and used to define the eigenvalue. Inverse iteration is the recommended method to obtain an eigenvector corresponding to a given eigenvalue.

Eigenvalue Bounds

There are two theorems due to Gerschgorin that define bounds for the eigenvalues of a matrix that are used in conjunction with other methods:-

Every eigenvalue of the matrix A lies in at least one of the circular discs with centres $a_{i,i}$ and radii $\sum_{j\neq i} |a_{i,j}|$.

If s of these discs form a connected domain which is isolated from the other discs, then there are precisely s eigenvalues of A within this connected domain.

Of particular importance is the use of these theorems with the symmetric matrix C, when the maximum and minimum bounds for all eigenvalues can be given as,

$$\min_i (c_{i,i} - \sum_{j\neq i} |c_{i,j}|) \leq \text{any eigenvalue of } C \leq \max_i (c_{i,i} + \sum_{j\neq i} |c_{i,j}|). \tag{2.4.27}$$

Sturm Sequences and Bisection

For a symmetric matrix with real eigenvalues the method of bisection can be used to determine the eigenvalues. Of more importance is the fact that bisection can be used to determine the p^{th} largest eigenvalue say, or the number of eigenvalues and their value in the range x to y (say). It is this ability to determine an eigenvalue independently of any others in the eigenvalue spectrum that makes the method so powerful and useful.

If C is the symmetric matrix,

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & c_{2,3} & \cdots & c_{2,N} \\ \vdots & \vdots & \vdots & & \vdots \\ c_{N,1} & c_{N,2} & c_{N,3} & \cdots & c_{N,N} \end{bmatrix} , \qquad (2.4.28)$$

where,

$$c_{i,j} = c_{j,i} . \qquad (2.4.29)$$

Then $P_r(\lambda)$, the leading principal minor of order r of the matrix $(C-\lambda I)$ is given by,

$$P_r(\lambda) = \det \begin{bmatrix} c_{1,1}-\lambda & c_{1,2} & c_{1,3} & \cdots & c_{1,r} \\ c_{2,1} & c_{2,2}-\lambda & c_{2,3} & \cdots & c_{2,r} \\ \vdots & \vdots & \vdots & & \vdots \\ c_{r,1} & c_{r,2} & c_{r,3} & \cdots & c_{r,r}-\lambda \end{bmatrix} . \qquad (2.4.30)$$

Obviously,

$$P_N(\lambda) = \det (C-\lambda I) , \qquad (2.4.31)$$

and $P_0(\lambda)$ is defined by,

$$P_0(\lambda) = 1 .$$

The sequence $P_i(\lambda)$, i=0,1,...,N is a sequence of polynomials in λ for which it can be shown that the zeros of $P_r(\lambda)$ strictly separate those of $P_{r-1}(\lambda)$. It is this property that provides the basis for the bisection method.

If the sequence $P_i(\lambda)$, i=0,1,...,N is evaluated for some value of λ then $s(\lambda)$ is the number of sign agreements between successive members of this sequence, (e.g. if $P_0(\lambda)=1$, $P_1(\lambda)=10$, $P_2(\lambda)=-1$, $P_3(\lambda)=-2$, $P_4(\lambda)=-4$, then $s(\lambda)=3$). If any $P_i(\lambda)$ is evaluated as zero it is taken to have the same sign as the preceeding member of the sequence $(P_{i-1}(\lambda))$.

The Sturm sequence property theorem can now be stated as:- The number of agreements in sign $s(\lambda)$ of successive members of the sequence $P_i(\lambda)$, i=0,1,...,N is equal to the number of eigenvalues of C which are strictly greater than λ.

The operation of the bisection method can now be described as used to obtain the k^{th} largest eigenvalue (λ_k) of the matrix C.

By the use of Gerschgorin's theorem two numbers a,b, can be obtained such that,

$$a < \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots\ldots\ldots \leq \lambda_N < b \ , \qquad (2.4.32)$$

where λ_i, i=1,2,...,N are the eigenvalues of C. Then the number $(a+b)/2=\mu_1$ is determined and the sequence $P_i(\mu)$, i=0,1,...,N calculated, from which can be determined $s(\mu_1)$. If $s(\mu_1)$ is greater than or equal to N-k then λ_k lies in the closed interval $[\mu_1,b]$, else λ_k lies in the closed interval $[a,\mu_1]$. The process is now repeated using whichever of the two half intervals λ_k lies in, with μ_2 being calculated as either $(\mu_1+b)/2$ or $(\mu_1+a)/2$. This process can now be repeated, always using the interval which λ_k lies in, until the width of the interval is small enough to obtain λ_k to the required accuracy. Wilkinson (1965) p.300.

Newton-Raphson Iteration and Secant Method

These two methods enable the eigenvalues of general matrices to be determined. They are both root finding methods for determining the zeros of a polynomial, and are therefore used to find the roots of the characteristic equation by working with the determinant of the matrix.

If matrix A is a general matrix then,

$$P_N(\lambda) = \det(A - \lambda I) \quad , \tag{2.4.33}$$

$$P'_N(\lambda) = \frac{d}{d\lambda} \det(A - \lambda I) \quad , \tag{2.4.34}$$

and $P_N(\lambda)$ is the value of the characteristic polynomial at λ, which is zero when λ is an eigenvalue of A.

The Newton-Raphson iteration can be defined as,

$$\lambda^{i+1} = \lambda^i - \frac{P_N(\lambda^i)}{P'_N(\lambda^i)} \quad , \quad (P'_N(\lambda) \neq 0), \tag{2.4.35}$$

where λ^1 is an initial guess at an eigenvalue. This method has the usual convergence properties (i.e. quadratic for single roots) that the method has when applied to ordinary polynomials. If the matrix has complex eigenvalues this method will locate them, but to do this λ^1 must be complex. One of the disadvantages of this method is that at each step both the determinant and its differential must be calculated. If for some matrices it is impracticable to determine $P'_N(\lambda)$ then in this case the Secant method can be used, which is defined by,

$$\lambda^{i+1} = \lambda^i - \frac{(\lambda^i - \lambda^{i-1})P_N(\lambda^i)}{(P_N(\lambda^i) - P_N(\lambda^{i-1}))} \quad , \tag{2.4.36}$$

which is the Newton-Raphson iteration with $P'_N(\lambda)$ replaced by its finite difference approximation

$$P'_N(\lambda^i) = \frac{(P_N(\lambda^i) - P_N(\lambda^{i-1}))}{(\lambda^i - \lambda^{i-1})} \quad . \tag{2.4.37}$$

This method cannot be used to find eigenvalues in the complex plane from a real initial guess (λ^1) so it is only used on symmetric matrices, or unsymmetric matrices where for instance the physical problem guarantees the existence of real

eigenvalues. Two previous determinant evaluations are required at each step, but starting the process to guarantee convergence can sometimes be a problem.

This leaves the problem of determining complex eigenvalues for matrices for which $P'_N(\lambda)$ cannot be calculated. This is achieved using Mullers' method which is described in Chapter 5.

With these methods when one eigenvalue has been found steps must be taken to avoid re-determining this same eigenvalue. This can be done simply by dividing $P_N(\lambda)$ by the difference of the current guess from each previously determined eigenvalue. The new function to be used if the ℓ eigenvalues $\lambda_1,\lambda_2,\ldots,\lambda_\ell$ have already been found is,

$$G_N(\lambda^i) = P_N(\lambda^i)/\prod_{j=1}^{\ell}(\lambda_j-\lambda^i) \quad . \tag{2.4.38}$$

$G_N(\lambda^i)$ can then be substituted in (2.4.34) and (2.4.35) to effectively deflate the matrix as each eigenvalue is found.

Transformation Methods

This section briefly describes some of the more popular transformation methods that can roughly be divided into two categories, those that obtain the complete solution, and those that transform the matrix to some simpler form to allow other efficient methods to be used (e.g. transformation to tridiagonal form, then use bisection).

Given's method consists of performing plane rotations on the matrix to reduce elements singly or in symmetric pairs to zero in a set order. This is a similarity transformation leaving the eigenvalues invariant, the plane rotations being performed by elementary unitary matrices. If the matrix is Hermitian, reduction to tri-diagonal form can be guaranteed in a finite number of steps, if the matrix is unsymmetric then the transformation is to Upper Hessenberg form.

A more efficient method is the reduction to the same

tridiagonal or upper Hessenberg form by the Householder method. This is a similarity transformation by elementary unitary Hermitian matrices that zero all the required elements in a column at one time (and the corresponding row for the symmetric case).

The Givens method can be more efficient on sparse matrices, as it is relatively easy with the method to avoid eliminating elements which are already zero. The problem with both methods is that little economy of space can be made when using either method on a computer, as it is almost impossible to avoid storing the full matrix no matter how sparse. Also if efficient determination of the eigenvectors is required, all the transformation matrices must be stored in some form to obtain the eigenvectors from those of the transformed matrix.

The LR transformation is a similarity transformation developed by Rutishauser (1958). This consists of decomposing the matrix (A say) to the form,

$$A = LR \quad , \tag{2.4.39}$$

where L is unit lower triangular and R is upper triangular. The similarity transform of A, $L^{-1}AL$ is defined by,

$$L^{-1}AL = L^{-1}(LR)L = RL \; . \tag{2.4.40}$$

If the original matrix is A_1 then the LR method can be written as

$$\left.\begin{aligned} A_{s-1} &= L_{s-1}R_{s-1} \quad , \\ A_s &= R_{s-1}L_{s-1} \quad . \end{aligned}\right\} \tag{2.4.41}$$

Rutishauser has shown that under certain conditions that as $s \to \infty$, $L_s \to I$, and R_s tends to an upper triangular matrix with the eigenvalues of A_1 situated on the main diagonal.

This method has a number of drawbacks, such as converging very slowly, and the decomposition breaking down. These have to some extent been combated by a shift of origin and interchanging rows, so that the modified algorithm is a powerful method.

The much more powerful QR algorithm was developed in 1961 by Francis. This is similar to the LR method, but instead of a triangular decomposition factorises the matrix into the product of a unitary matrix Q, and an upper triangular matrix R. The algorithm can be defined by the equations,

$$A_s = Q_s R_s \quad , \quad A_{s+1} = Q_s^H A_s Q_s = Q_s^H Q_s R_s Q_s = R_s Q_s \quad , \qquad (2.4.42)$$

where A_1 is the original matrix. Obviously it is a similarity transformation and A_s has the same eigenvalues as A_1.

For each step the QR algorithm requires more work than the LR algorithm, but there is a better guarantee of convergence, which is more rapid. Even so, a shift of origin can again be introduced to speed convergence.

Because of practical considerations of work load and difficulites with the factorisation, the QR method is only used on upper Hessenberg or symmetric band matrices. When this method is programmed on a computer, even for large matrices the storage problems are not too excessive. The method is then the quickest available to obtain all the eigenvalues of the matrix in question. However in practice only a few eigenvalues of the matrix may be required allowing other methods to compete on a time basis. It is for this reason that methods developed in later chapters are compared whenever possible with QR and LR methods.

CHAPTER 3

METHODS FOR DETERMINING THE EIGENVALUES AND EIGENVECTORS OF PERIODIC TRIDIAGONAL MATRICES

3.1 INTRODUCTION

This Chapter is concerned with finding the eigenvalues of the periodic tridiagonal matrix. This matrix occurs for example in the finite difference approximation to the Sturm-Liouville differential equation with periodic boundary conditions, the modal analysis of Floquet waves in a composite material, and other applications particularly with periodic boundary conditions.

The periodic characteristic Sturm-Liouville problem can be defined by,

$$\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y + \lambda_r(x)y = 0 \ , \tag{3.1.1}$$

where the numerical values of λ and $y(x)$ are required over the range $[a,b]$, with the boundary conditions,

$$\left.\begin{aligned} y(a) &= y(b) \quad , \\ p(a)\frac{dy}{dx}(a) &= p(b)\frac{dy}{dx}(b) \ . \end{aligned}\right\} \tag{3.1.2}$$

The direct substitution for the second derivative in (3.1.1) by the approximation

$$\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) \simeq \frac{p_{i+\frac{1}{2}}(y_{i+1}-y_i)-p_{i-\frac{1}{2}}(y_i-y_{i-1})}{h^2} \ , \tag{3.1.3}$$

at each of the discrete points x_i, $i=1,2,\ldots,N$, in the interval $[a,b]$ where $Nh=b-a$, yields homogeneous linear equations of the form,

$$Ay = \lambda h^2 Ry \quad , \tag{3.1.4}$$

where A is the matrix

$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & & a_{1,N} \\ a_{2,1} & a_{2,2} & a_{2,3} & & 0 & \\ & a_{3,2} & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & a_{N-1,N} \\ a_{N,1} & & & a_{N,N-1} & a_{N,N} & \end{bmatrix} \ , \tag{3.1.5}$$

with elements,

$$\left.\begin{aligned} a_{i,i} &= (p_{i+\frac{1}{2}} + p_{i-\frac{1}{2}}) - q_i h^2 \ , \ i=1,2,\ldots,N, \\ a_{i,i+1} &= -p_{i+\frac{1}{2}}, \ a_{i,i-1} = -p_{i-\frac{1}{2}}, \ i=1,2,\ldots,N \ , \end{aligned}\right\} \qquad (3.1.6)$$

where $a_{1,N}=a_{1,0}$, $a_{N,1}=a_{N,N+1}$.

The matrix R is a diagonal matrix with elements f_i, i=1,2,...,N, all greater than zero. Therefore equation (3.1.4) can be written,

$$C\underline{u} = \lambda\underline{u} \ , \qquad (3.1.7)$$

where

$$\left.\begin{aligned} \underline{y} &= R^{-\frac{1}{2}}\underline{u} \ , \\ C &= h^2 R^{-\frac{1}{2}} A R^{-\frac{1}{2}} \ . \end{aligned}\right\} \qquad (3.1.8)$$

The matrix C has the same form as A, and equation (3.1.7) represents the standard eigenvalue problem to be solved.

A new formulation of the Sturm sequence for the symmetric periodic tridiagonal matrix is discussed in detail and its equivalence to previously obtained sequences is shown. This formulation of the Sturm sequence is then extended to obtain the sequence for unsymmetric matrices. The Sturm sequence is then used in an implicit Bairstow technique to find the eigenvalues of an unsymmetric matrix. Finally the sequence is used in a Newton type iteration for both symmetric and unsymmetric matrices and results are compared with the previously described methods.

3.2 DETERMINATION OF THE STURM SEQUENCE FOR A SYMMETRIC PERIODIC TRIDIAGONAL MATRIX

Let C be a symmetric periodic (N×N) tridiagonal matrix derived from the finite difference discretisation of a Sturm-Liouville system as indicated in the previous section. In order to find the eigenvalues of C a solution must be found to the determinantal equation

$$\det(C-\lambda I) = 0 \ , \qquad (3.2.1)$$

where,

$$|C-\lambda I| = \det \begin{bmatrix} c_1-\lambda & b_2 & & & b_1 \\ b_2 & c_2-\lambda & b_3 & & \\ & b_3 & \ddots & \ddots & \\ & & \ddots & \ddots & b_N \\ b_1 & & & b_N & c_N-\lambda \end{bmatrix} = 0 \qquad (3.2.2)$$

Evans (1971) has shown that the characteristic polynomials of the matrix $(C-\lambda I)$ are given by the following relationships,

$$\left.\begin{aligned} P_{-1}(\lambda) &= 0 \quad , \\ P_0(\lambda) &= 1 \quad , \\ P_1(\lambda) &= c_1-\lambda \quad , \\ P_2(\lambda) &= (c_2-\lambda)P_1(\lambda) - b_2^2 P_0(\lambda) \quad , \\ &\vdots \\ P_i(\lambda) &= (c_i-\lambda)P_{i-1}(\lambda) - b_i^2 P_{i-2}(\lambda), \quad i=3,\ldots,N-1, \end{aligned}\right\} \qquad (3.2.3)$$

and

$$\left.\begin{aligned} Q_0(\lambda) &= 0 \quad , \\ Q_1(\lambda) &= 1 \quad , \\ Q_2(\lambda) &= (c_2-\lambda) \quad , \\ &\vdots \\ Q_i(\lambda) &= (c_i-\lambda)Q_{i-1}(\lambda) - b_i^2 Q_{i-2}(\lambda), \quad i=3,\ldots,N-1, \end{aligned}\right\} \qquad (3.2.4)$$

Finally,

$$P_N(\lambda) = (c_N-\lambda)P_{N-1}(\lambda) - b_N^2 P_{N-2}(\lambda) - b_1^2 Q_{N-1}(\lambda) + (-1)^{N-1} 2 \prod_{j=1}^{N} b_j . \qquad (3.2.5)$$

These polynomials $(P_i(\lambda), i=1,N)$ were obtained from a Laplace expansion of the matrix $(C-\lambda I)$, and are the leading principal minors of $|C-\lambda I|$. Hohn (1964) These polynomials can now be used in a bisection process to determine the eigenvalues of the matrix C. Unfortunately even for small well behaved matrices the polynomials $P_i(\lambda)$, $i=1,2,\ldots,N$ oscillate wildly between large positive and negative values, even for values of λ

quite close to an eigenvalue. In an attempt to avoid underflow and overflow occurring when calculating the sequence in floating point arithmetic on a computer, the procedure of Barth et al., (1967) can be adopted.

The sequence of polynomials $P_i(\lambda)$ and $Q_i(\lambda)$ are replaced by a new sequence of scaled polynomials $p_i(\lambda)$ and $q_i(\lambda)$ given by,

$$p_i(\lambda) = P_i(\lambda)/P_{i-1}(\lambda) \quad , \tag{3.2.6}$$

and

$$q_i(\lambda) = Q_i(\lambda)/P_i(\lambda) \quad , \tag{3.2.7}$$

The relationships (3.2.3), (3.2.4), and (3.2.5) then become,

$$\left.\begin{aligned} p_0(\lambda) &= 1 \quad , \\ p_i(\lambda) &= (c_1-\lambda) \quad , \\ p_2(\lambda) &= c_2-\lambda-b_2^2/p_1(\lambda) \quad , \\ &\vdots \\ p_i(\lambda) &= c_i-\lambda-b_i^2/p_{i-1}(\lambda) \quad , \quad i=3,4,\ldots,N-1, \end{aligned}\right\} \tag{3.2.8}$$

whilst the $q_i(\lambda)$ after some simple algebraic manipulation become,

$$\left.\begin{aligned} q_1(\lambda) &= 1 \\ q_2(\lambda) &= (c_2-\lambda)/p_1(\lambda) \\ q_3(\lambda) &= (c_3-\lambda)\, q_2(\lambda)/p_3(\lambda)-b_3^2 q_1(\lambda)/(p_2(\lambda)p_3(\lambda)), \\ &\vdots \\ q_i(\lambda) &= (c_i-\lambda)q_{i-1}(\lambda)/p_i(\lambda)-b_i^2 q_{i-2}(\lambda)/(p_i(\lambda)p_{i-1}(\lambda)), \\ & \qquad\qquad i=3,4,\ldots,N \, , \end{aligned}\right\} \tag{3.2.9}$$

and then finally,

$$p_N(\lambda) = (c_N-\lambda)-b_N^2/p_{N-1}(\lambda)-b_1^2 q_{N-1}(\lambda)+ 2b_N \prod_{j=1}^{N-1}-(b_j/p_j(\lambda)). \tag{3.2.10}$$

Overflow and underflow are now avoided whilst calculating the $p_i(\lambda)$ (i=1,N) and instead of counting the sign changes in $P_i(\lambda)$ as in the previous procedure (3.2.3) and (3.2.5), the number of negative signs in the sequence $p_i(\lambda)$ are now counted. This indicates the number of eigenvalues less than λ and can be used in the usual bisection process in the same way, as a negative $p_i(\lambda)$ indicates $P_i(\lambda)$ & $P_{i-1}(\lambda)$ are of a different sign.

An equivalent sequence to (3.2.8), (3.2.9), and (3.2.10) can be obtained in a different manner to the Laplace expansion method used above. The sequence obtained is *more efficient* to compute than (3.2.8-3.2.10), and the method by which it is obtained forms the basis for much of the ensuing analysis. For this reason the method is described in detail.

If a Gaussian elimination procedure is performed on the matrix $(C-\lambda I)$ the matrix will be transformed to upper triangular form in the following stages.

By setting

$$(c_1-\lambda) = s_1(\lambda), \qquad r_1 = b_1 \ , \tag{3.2.11}$$

then for the first step the matrix becomes,

$$\begin{bmatrix} s_1(\lambda), & b_2 & & & & r_1 \\ 0 & , c_2-\lambda-b_2^2/s_1(\lambda), & b_3 & & & -r_1b_2/s_1(\lambda) \\ & b_3 & c_3-\lambda & \ddots & 0 & \\ & & \ddots & \ddots & \ddots & \\ & & 0 & \ddots & \ddots & b_N \\ 0 & , -r_1b_2/s_1(\lambda) & & & b_N, & c_N-\lambda-r_1^2/s_1(\lambda) \end{bmatrix} \tag{3.2.12}$$

Again setting

$$c_2-\lambda- b_2^2/s_1(\lambda) = s_2(\lambda), \; r_2 = -r_1b_2/s_1(\lambda), \tag{3.2.13}$$

the second step of the Gaussian elimination process becomes,

$$\begin{bmatrix} s_1(\lambda) & b_2 & & & & r_1 \\ 0 & s_2(\lambda) & b_3 & & & r_2 \\ & 0 & c_3-\lambda-b_3^2/s_2(\lambda), & b_4 & & -r_2b_3/s_2(\lambda) \\ & & b_4 & \ddots & 0 & \\ & & & \ddots & \ddots & b_N \\ 0 & 0 & -r_2b_3/s_2(\lambda) & & b_N & c_N-\lambda-r_1^2/s_1(\lambda)-r_2^2/s_2(\lambda) \end{bmatrix} \tag{3.2.14}$$

This elimination process is repeated for (N-1) steps and it can easily be seen that the final form of the matrix then becomes,

$$\begin{bmatrix} s_1(\lambda) & b_2 & & & & r_1 \\ & s_2(\lambda) & b_3 & & & r_2 \\ & & s_3(\lambda) & & 0 & \\ & & & \ddots & & \\ & & 0 & & & b_N \\ & & & & & s_N(\lambda) \end{bmatrix} r_{N-1} \quad , \tag{3.2.15}$$

where,

$$\left.\begin{aligned} s_0(\lambda) &= 1 \quad , \\ s_1(\lambda) &= c_1-\lambda \quad , \\ s_2(\lambda) &= c_2-\lambda-b_2^2/s_1(\lambda) \quad , \\ &\vdots \\ s_i(\lambda) &= c_i-\lambda-b_i^2/s_{i-1}(\lambda) \quad , \quad i=3,4,\ldots,N-1, \\ r_1 &= b_1 \\ r_i &= -r_{i-1}b_i/s_{i-1}(\lambda), \quad i=2,3,\ldots,N-1, \end{aligned}\right\} \tag{3.2.16}$$

and finally,

$$s_N(\lambda) = c_N-\lambda-\sum_{j=1}^{N-1} r_j^2/s_j(\lambda)-2\, b_N r_{N-1}/s_{N-1}(\lambda)-b_N^2/s_{N-1}(\lambda) \quad .$$

The sequence $s_i(\lambda)$, i=1,N is identical to the sequence $p_i(\lambda)$, i=1,N, and their equivalence will now be shown in the following manner.

If (3.2.16) and (3.2.3) are compared it is seen that,

$$s_1(\lambda) = \frac{P_1(\lambda)}{P_0(\lambda)} = c_1-\lambda \quad , \tag{3.2.17}$$

which on substitution in $s_2(\lambda)$ gives,

$$s_2(\lambda) = c_2-\lambda-b_2^2/\left(\frac{P_1(\lambda)}{P_0(\lambda)}\right) \quad , \tag{3.2.18}$$

and on clearing terms becomes,

$$P_1(\lambda)s_2(\lambda) = (c_2-\lambda)P_1(\lambda) -b_2^2P_0(\lambda) \quad . \tag{3.2.19}$$

Then by comparing (3.2,19) and (3,2,3), we have,

$$P_1(\lambda)s_2(\lambda) = P_2(\lambda) \quad , \tag{3.2.20}$$

and therefore,

$$s_2(\lambda) = P_2(\lambda)/P_1(\lambda) \quad . \tag{3.2.21}$$

If this procedure is continued it can be seen that,

$$s_i(\lambda) = \frac{P_i(\lambda)}{P_{i-1}(\lambda)} = p_i(\lambda), \quad i=1,2,\ldots,N-1, \tag{3.2.22}$$

and,

$$P_i(\lambda) = \prod_{j=1}^{i} s_j(\lambda), \qquad i=1,2,\ldots,N-1. \tag{3.2.23}$$

From (3.2.16) and (3.2.10) for $p_N(\lambda)$ to be equal to $s_N(\lambda)$ the following relationships must be true,

$$b_1^2 q_{N-1}(\lambda) + 2\, b_N \prod_{j=1}^{N-1} (-b_j/p_i(\lambda)) = \sum_{j=1}^{N-1} \frac{r_j^2}{s_j(\lambda)} - 2b_N r_{N-1}/s_{N-1}(\lambda) \tag{3.2.24}$$

Using continual substitution of the relationship from (3.2.16) describing the r_i,

$$\left.\begin{aligned} \frac{-2b_N r_{N-1}}{s_{N-1}(\lambda)} &= \frac{2b_N\, b_{N-1} r_{N-2}}{s_{N-1}(\lambda)s_{N-2}(\lambda)} \\ &= \frac{-2b_N\, b_{N-1}\, b_{N-2} r_{N-3}}{s_{N-1}(\lambda)s_{N-2}(\lambda)s_{N-3}(\lambda)} \\ &\;\;\vdots \\ &= 2b_N \sum_{i=1}^{N-1} (-b_i/s_i(\lambda)) \end{aligned}\right\} \tag{3.2.25}$$

Substituting for the s_i from (3.2.22)

$$\frac{-2b_N r_{N-1}}{s_{N-1}(\lambda)} = 2b_N \prod_{i=1}^{N-1} (-b_i/p_i(\lambda)) \quad . \tag{3.2.26}$$

Therefore subtracting relationship (3.2.26) from (3.2.24) it only remains to show that,

$$b_1^2\, q_{N-1}(\lambda) = \sum_{i=1}^{N-1} r_i^2/s_i(\lambda) \quad , \tag{3.2.27}$$

or by using equation (3.2.7),

$$b_1^2 \, Q_{N-1}(\lambda)/P_{N-1}(\lambda) = \sum_{i=1}^{N-1} r_i^2/s_i(\lambda) \quad . \tag{3.2.28}$$

This is achieved by an induction proof.

Assume that

$$b_1^2 \, Q_{M-1}(\lambda)/P_{M-1}(\lambda) = \sum_{i=1}^{M-1} r_i^2/s_i(\lambda) \quad , \tag{3.2.29}$$

and that,

$$b_1^2 \, Q_{M-2}(\lambda)/P_{M-2}(\lambda) = \sum_{i=1}^{M-2} r_i^2/s_i(\lambda) \quad . \tag{3.2.30}$$

Using the recurrence relationship of (3.2.4) $Q_M(\lambda)$ can be written as

$$Q_M(\lambda) - (c_M-\lambda)Q_{M-1}(\lambda) - b_M^2 \, Q_{M-2}(\lambda) \ ,$$

and it therefore follows that,

$$b_1^2 Q_M(\lambda)/P_M(\lambda) = (c_M-\lambda)Q_{M-1}(\lambda)b_1^2/P_M(\lambda) - b_M^2 b_1^2 Q_{M-1}(\lambda)/P_M(\lambda) \ . \tag{3.2.31}$$

Now by substituting for $Q_{M-1}(\lambda)$ and $Q_{M-2}(\lambda)$ from equations (3.2.28) and (3.2.29) the relationship (3.2.31) becomes,

$$b_1^2 Q_M(\lambda)/P_M(\lambda) = \frac{(c_M-\lambda)P_{M-1}(\lambda)}{P_M(\lambda)} \sum_{i=1}^{M-1} \frac{r_i^2}{s_i(\lambda)} - \frac{b_M^2 P_{M-2}(\lambda)}{P_M(\lambda)} \sum_{i=1}^{M-1} \frac{r_i^2}{s_i(\lambda)} \ . \tag{3.2.32}$$

Next the $p_i(\lambda)$ can be replaced by all the equivalent $s_i(\lambda)$ as given in (3.2.22), and $(c_M-\lambda)$ can be replaced using the relationship (3.2.16), then (3.2.32) becomes,

$$b_1^2 Q_M(\lambda)/P_M(\lambda) = \frac{1}{s_M(\lambda)} \left(s_M(\lambda) + \frac{b_M^2}{s_{M-1}(\lambda)}\right) \left(\sum_{i=1}^{M-1} \frac{r_i^2}{s_i(\lambda)}\right) - b_M^2 \left(\sum_{i=1}^{M-2} \frac{1}{s_M(\lambda)s_{M-1}(\lambda)} \frac{r_i^2}{s_i(\right.$$

(3.2.33)

$$= \sum_{i=1}^{M-1} \frac{r_i^2}{s_i(\lambda)} + \frac{r_M}{s_M(\lambda)} + \frac{b_M^2}{s_M(\lambda)s_{M-1}(\lambda)} \left(\sum_{i=1}^{M-2} \frac{r_i^2}{s_i(\lambda)}\right)$$

$$- \frac{b_M^2}{s_M(\lambda)s_{M-1}(\lambda)} \left(\sum_{i=1}^{M-2} \frac{r_i^2}{s_i(\lambda)}\right) \quad . \tag{3.2.34}$$

Now the two terms of opposite sign cancel and all remaining terms can be placed in the sum to leave,

$$b_1^2 Q_M(\lambda)/P_M(\lambda) = \sum_{i=1}^{M} r_i^2/s_i(\lambda) \quad . \tag{3.2.35}$$

This shows that if relationships (3.2.29) and (3.2.30) are true for any M-1, M-2, then (3.2.35) is true for any M.

If M-2 equals 1 then the L.H.S. of equation (3.2.30) becomes,

$$\frac{b_1^2\, Q_1(\lambda)}{P_1(\lambda)} = \frac{b_1^2}{(c_1-\lambda)} \quad , \tag{3.2.36}$$

by substituting from equations (3.2.4) and (3.2.3). The right hand side of equation (3.2.30) after substituting from (3.2.16) becomes,

$$\frac{r_1^2}{s_1(\lambda)} = \frac{b_1^2}{(c_1-\lambda)} \quad , \tag{3.2.37}$$

Therefore from equations (3.2.36) and (3.2.37) equation (3.2.30) is shown true when M-2 has the value 1.

If M-1 equals 2 then the L.H.S. of equation (3.2.29) after substituting from equations (3.2.3) and (3.2.4) becomes,

$$\frac{b_1^2\, Q_2(\lambda)}{P_2(\lambda)} = \frac{b_1^2\,(c_2-\lambda)}{(c_2-\lambda)P_1-b_2^2} = \frac{b_1^2(c_2-\lambda)}{(c_2-\lambda)(c_1-\lambda)-b_2^2} \quad . \tag{3.2.38}$$

By taking the R.H.S. of equation (3.2.30) and substituting using (3.2.16) the expression becomes,

$$\begin{aligned}
\sum_{i=1}^{2} \frac{r_i^2}{s_i(\lambda)} &= \frac{r_1^2}{s_1(\lambda)} + \frac{r_2^2}{s_2(\lambda)} \\
&= \frac{b_1^2}{s_1(\lambda)} + \frac{b_1^2 b_2^2}{s_1^2(\lambda)s_2(\lambda)} \\
&= \frac{b_1^2}{(c_1-\lambda)}\left(1 + \frac{b_2^2}{s_1(\lambda)s_2(\lambda)}\right) \\
&= \frac{b_1^2}{(c_1-\lambda)}\left(\frac{(c_2-\lambda)(c_1-\lambda)-b_2^2+b_2^2}{(c_1-\lambda)\left(c_2-\lambda-\frac{b_2^2}{(c_1-\lambda)}\right)}\right) \\
&= \frac{b_1^2\,(c_2-\lambda)}{(c_1-\lambda)(c_2-\lambda)-b_2^2} \quad .
\end{aligned} \tag{3.2.39}$$

Therefore from equations (3.2.38) and (3.2.39) equation (3.2.29) is shown true when M-1 has the value 2.

Therefore (3.2.29) and (3.2.30) have been shown true for M-1,M-2, equal to 2,1, and equation (3.2.27) is proved true by induction for all M.

It has now been proved that,

$$s_i(\lambda) = \frac{P_i(\lambda)}{P_{i-1}(\lambda)} = p_i(\lambda) \quad , \qquad i=1,N \quad , \tag{3.2.40}$$

for any symmetric periodic tridiagonal matrix. In fact a similar relationship can be proved for any unsymmetric periodic tridiagonal matrix in the same manner.

The sequence $p_i(\lambda)$, i=1,N can now be replaced by the sequence $s_i(\lambda)$, i=1,N and used in a bisection process to isolate the eigenvalues of the matrix C as described in Chapter 2.

3.3 RESULTS

The recurrence relationship (3.2.16) was programmed in ALGOL 60 on the I.C.L. 1904S computer at Loughborough University of Technology. The procedure is given as program 2 in Appendix 1.

The program was tested on the following 11×11 matrix,

$$\begin{bmatrix} 2 & -1 & & & & & & & & & -1 \\ -1 & 2 & -1 & & & & & & & & \\ & -1 & 2 & -1 & & & & & & & \\ & & \ddots & \ddots & \ddots & & & & & & \\ & & & \ddots & \ddots & \ddots & & & & & \\ & & & & \ddots & \ddots & \ddots & & & & \\ & & & & & \ddots & \ddots & \ddots & & & \\ & & & & & & \ddots & \ddots & -1 & & \\ & & & & & & & -1 & 2 & & \\ -1 & & & & & & & & & & \end{bmatrix} \tag{3.3.1}$$

The eigenvalues are known and are given by,

$$\lambda_r = 4 \sin^2 (r\pi/N) \quad , \tag{3.3.2}$$

the theoretical and calculated eigenvalues are tabulated and compared in Table 3.3.1, the results being rounded to 10 significant figures

i	Theoretical λ_i	Actual results $\bar{\lambda}_i$	$\lambda_i - \bar{\lambda}_i$
1	$2.168404345 \times 10^{-19}$	$1.682132106 \times 10^{-11}$	$-1.682132084 \times 10^{-11}$
2	$3.174929341 \times 10^{-1}$	$3.174929343 \times 10^{-1}$	-0.000000002
3	$3.174929341 \times 10^{-1}$	$3.174929343 \times 10^{-1}$	0.000000002
4	1.169169974	1.169169974	0.000000000
5	1.169169974	1.169169974	0.000000000
6	2.284629676	2.284629677	-0.000000001
7	2.284629677	2.284629677	0.000000000
8	3.309721468	3.309721468	0.000000000
9	3.309721468	3.309721468	0.000000000
10	3.918985947	3.918985947	0.000000000
11	3.918985947	3.918985947	0.000000000
		$\sqrt{\sum_{i=1}^{11} (\lambda_i - \bar{\lambda}_i)^2}$	0.000000003

TABLE 3.3.1

The computer took 2.5 seconds to obtain the results given in Table (3.3.1), and they can be seen to be accurate to 9 decimal places.

Next the solution of a larger problem was sought in which the resulting matrix is of order 60×60. The theoretical and experimental values of the ten largest eigenvalues are given in Table 3.3.2.

i	Theoretical λ_i	Actual results $\bar{\lambda}_i$	$\sqrt{(\lambda_i - \bar{\lambda}_i)^2}$
1	4.000000000	4.000000000	0.000000000
2	3.984229403	3.984229410	0.000000007
3	3.984229403	3.984229403	0.000000000
4	3.937166322	3.937166330	0.000000008
5	3.937166322	3.937166308	0.000000014
6	3.859552972	3.859553001	0.000000029
7	3.859552972	3.859552951	0.000000021
8	3.752613360	3.752613370	0.000000010
9	3.752613360	3.752613359	0.000000001
10	3.618033989	3.618034000	0.000000021

TABLE 3.3.2

The accuracy of these results has been reduced to 8 significant figures. This is due partly to the size of the matrix, and because all the eigenvalues except the largest occur in double pairs, and accuracy is always reduced when finding multiple and close eigenvalues.

The results are compared with a N.A.G. library routine using an L.R. procedure to obtain the eigenvalues. The results obtained by the library routine agreed with the theoretical results to 10 significant figures, and were obtained in 75% of the time taken by program 2 (5 seconds). However the workspace required by the N.A.G. library routine in the computer for storing arrays was $O(N^2)$ whereas the storage used by program 2 was O(N). It is interesting to note that neither program was able to obtain the roots as double roots, all were given as pairs of very close roots.

As a final test program 2 was run on a 300×300 matrix of the same form as (3.3.1) to obtain the largest and smallest eigenvalues. The answers (4.000000000, and 0.000000000) were obtained correct to 10 significant figures in 11 seconds.

3.4 THE APPLICATION OF BAIRSTOWS METHOD TO FIND THE EIGENVALUES OF AN UNSYMMETRIC PERIODIC TRIDIAGONAL MATRIX

Bairstows method is a procedure for finding the real quadratic factors of a given polynomial, thus determining the roots of the polynomial in real or complex conjugate pairs. This method is applied to matrices by finding quadratic factors of the characteristic polynomial implicitly, and the eigenvalues are then obtained in pairs as roots of the quadratic factors.

Let D be an unsymmetric periodic tridiagonal matrix of order N then, the eigenvalues are given by,

$$|D-\lambda I| = \det \begin{bmatrix} c_1-\lambda & b_2 & & & d_1 \\ d_2 & c_2-\lambda & b_3 & & \\ & d_3 & \ddots & \ddots & \\ & & \ddots & \ddots & b_N \\ b_1 & & & d_N & c_N-\lambda \end{bmatrix} = 0 \qquad (3.4.1)$$

It can easily be seen, by using (3.2.3),(3.2.4), and (3.2.5) that the Sturm sequence for the matrix (3.4.1) is given by,

$$\left.\begin{aligned} P_{-1}(\lambda) &= 0 \quad , \\ P_0(\lambda) &= 1 \quad , \\ P_1(\lambda) &= c_1-\lambda \quad , \\ P_2(\lambda) &= (c_2-\lambda)P_1(\lambda) - b_2d_2P_0(\lambda) \quad , \\ &\vdots \\ P_i(\lambda) &= (c_i-\lambda)P_{i-1}(\lambda) - b_id_iP_{i-2}(\lambda), \quad i=3,4,\ldots,N-1, \end{aligned}\right\} \qquad (3.4.2)$$

and

$$\left.\begin{aligned} Q_0(\lambda) &= 0 \quad , \\ Q_1(\lambda) &= 1 \quad , \\ Q_2(\lambda) &= (c_2-\lambda) \quad , \\ Q_3(\lambda) &= (c_3-\lambda)Q_2(\lambda) - b_3d_3Q_1(\lambda) \quad , \\ &\vdots \\ Q_i(\lambda) &= (c_i-\lambda)Q_{i-1}(\lambda) - b_id_iQ_{i-2}(\lambda), \quad i=4,5,\ldots,N-1 \end{aligned}\right\} \qquad (3.4.3)$$

and finally,

$$P_N(\lambda) = (c_N-\lambda)P_{N-1}(\lambda) - b_Nd_NP_{N-2}(\lambda) - b_1d_1Q_{N-1}(\lambda) + (-1)^{N-1}\prod_{j=1}^{N} b_j + (-1)^{N-1}\prod_{j=1}^{N} d_j \; . \qquad (3.4.4)$$

Since the eigenvalues could be complex the Sturm sequence cannot be used in a bisection process. Instead Bairstows method will be used. As the method is quite difficult a slightly simpler

matrix than D will be used as an example to illustrate the algorithm. This facilitates explanation of the method, and extending the result to cater for D is a trivial step.

The matrix C is obtained by setting,

$$b_i = d_i, \quad i=2,N, \quad \text{and } a = b_1, \quad ad = d_1 , \tag{3.4.5}$$

in (3.4.1) so that the matrix is now of the form,

$$(C-\lambda I) = \begin{bmatrix} c_1-\lambda & b_2 & & & a \\ b_2 & c_2-\lambda & b_3 & & \\ & b_3 & c_3-\lambda & \ddots & \\ & & \ddots & \ddots & b_N \\ ad & & & b_N & c_N-\lambda \end{bmatrix} . \tag{3.4.6}$$

Then from (3.4.2), (3.4.3), and (3.4.4) it can be seen that

$$\left.\begin{aligned} P_{-1}(\lambda) &= 0 \quad , \\ P_0(\lambda) &= 1 \quad , \\ P_1(\lambda) &= c_1-\lambda \quad , \\ P_2(\lambda) &= (c_2-\lambda)P_1(\lambda) - b_2^2 P_0(\lambda) \quad , \\ &\vdots \\ P_i(\lambda) &= (c_i-\lambda)P_{i-1}(\lambda) - b_i^2 P_{i-2}(\lambda), \quad i=3,\ldots,N, \end{aligned}\right\} \tag{3.4.7}$$

and

$$\left.\begin{aligned} Q_0(\lambda) &= 0 \\ Q_1(\lambda) &= 1 \\ Q_2(\lambda) &= (c_2-\lambda) \\ &\vdots \\ Q_i(\lambda) &= (c_i-\lambda)Q_{i-1}(\lambda) - b_i^2 Q_{i-2}(\lambda), \quad i=3,\ldots,N-1, \end{aligned}\right\} \tag{3.4.8}$$

and finally,

$$P_N(\lambda) = (c_N-\lambda)P_{N-1}(\lambda) - b_N^2 P_{N-2}(\lambda) - ad.a.Q_{N-1}(\lambda) + (-1)^{N-1}(ad+a)\prod_{j=2}^{N} b_j \quad . \tag{3.4.9}$$

If each of the polynomials $P_i(\lambda)$ i=1,N from equation (3.4.7) and (3.4.9) are divided by a trial quadratic factor of the form $(\lambda^2-E\lambda-F)$ say, then a linear remainder, $(A_i\lambda+B_i$, i=1,N) say, is produced. The polynomials $P_i(\lambda)$ can now be written as,

$$P_i(\lambda) = (\lambda^2-E\lambda-F)R_i(\lambda) + A_i\lambda + B_i, \qquad i=1,N \ , \tag{3.4.10}$$

where $R_i(\lambda)$, i=1,N is a polynomial in λ of degree (i-2). Similarly each of the polynomials $Q_i(\lambda)$, i=1,N-1, from (3.4.8) are divided by the trial quadratic factor $(\lambda^2-E\lambda-F)$ producing a linear remainder, which is $G_i\lambda + H_i$, i=1,N-1. The polynomials $Q_i(\lambda)$ can now be written as,

$$Q_i(\lambda) = (\lambda^2-E\lambda-F)S_i(\lambda) + G_i\lambda + H_i, \qquad i=1,N-1. \tag{3.4.11}$$

Again the $S_i(\lambda)$, i=1,N-1 are polynomials in λ of degree (i-2). Now equations (3.4.10) and (3.4.11) are substituted into (3.4.9) to give,

$$\left.\begin{aligned}(\lambda^2-E\lambda-F)R_N(\lambda)+A_N\lambda+B_N \;=\; & (c_N-\lambda)((\lambda^2-E\lambda-F)R_{N-1}(\lambda)+A_{N-1}\lambda+B_{N-1}) \\ & -b_N^2\,((\lambda^2-E\lambda-F)R_{N-2}(\lambda)+A_{N-2}\lambda+B_{N-2}) \\ & -a.ad((\lambda^2-E\lambda-F)S_{N-1}(\lambda)+G_{N-1}\lambda+H_{N-1}) \\ & +(-1)^{N-1}(a+ad)\prod_{i=2}^{N} b_i \ .\end{aligned}\right\} \tag{3.4.12}$$

Next the coefficients of $(\lambda^2-E\lambda-F)$, λ^1 and λ^0 are equated from both sides of equation (3.4.12), and the result after some algebraic manipulation becomes,

$$\left.\begin{aligned}R_N(\lambda) &= (c_N-\lambda)R_{N-1}(\lambda) - b_N^2R_{N-2}(\lambda) - a.ad\ S_{N-1}-A_{N-1}, \\ A_N &= c_NA_{N-1}-A_{N-1}E-B_{N-1}-b_N^2A_{N-2} - a.ad\ G_{N-1}\ , \\ B_N &= c_NB_{N-1}-A_{N-1}F-b_N^2B_{N-2} - a.ad\ H_{N-1}+(-1)^{N-1}(a+ad)\prod_{i=2}^{N} b_i\end{aligned}\right\} \tag{3.4.13}$$

Now the $P_i(\lambda)$ and $Q_i(\lambda)$ can be substituted from equations (3.4.10) and (3.4.11) in equations (3.4.7) and (3.4.8). This produces, for the general term,

$$\left.\begin{aligned}(\lambda^2-E\lambda-F)R_i(\lambda)+A_i\lambda+B_i &= (c_i-\lambda)((\lambda^2-E\lambda-F)R_{i-1}(\lambda)+A_{i-1}\lambda+B_{i-1})\\ &\quad -b_i^2((\lambda^2-E\lambda-F)R_{i-2}(\lambda)+A_{i-2}\lambda+B_{i-1}),\\ (\lambda^2-E\lambda-F)S_i(\lambda)+G_i\lambda+H_i &= (c_i-\lambda)((\lambda^2-E\lambda-F)S_{i-1}(\lambda)+G_{i-1}\lambda+H_{i-1})\\ &\quad -b_i^2((\lambda^2-E\lambda-F)S_{i-2}(\lambda)+G_{i-2}\lambda H_{i-2}),\end{aligned}\right\} \quad i=1,\ldots,N-1, \tag{3.4.14}$$

Again the coefficients of $(\lambda^2-E\lambda-F)$, λ^1 and λ^0 are equated from both sides of equation (3.4.14), and the result after some trivial algebraic manipulation is,

$$\begin{aligned}R_i(\lambda) &= (c_i-\lambda)\,R_{i-1}(\lambda) - A_{i-1} - b_i^2\,R_{i-2}(\lambda)\ ,\\ A_i &= (c_i-E)A_{i-1}-B_{i-1}-b_i^2A_{i-2}\ ,\\ B_i &= c_iB_{i-1}-A_{i-1}F-b_i^2B_{i-2}\ ,\\ S_i(\lambda) &= (c_i-\lambda)S_{i-1}(\lambda) - G_{i-1} - b_i^2\,S_{i-2}(\lambda)\ ,\\ G_i &= (c_i-E)G_{i-1}-H_{i-1}-b_i^2G_{i-2}\ ,\\ H_i &= c_iH_{i-1}-G_{i-1}F-b_i^2H_{i-2}\ ,\end{aligned} \tag{3.4.15}$$

If it is noted that the starting values for the sequences $P_i(\lambda)$, and $Q_i(\lambda)$ from equations (3.4.7) and (3.4.8) are,

$$P_{-1}(\lambda) = 0,\quad P_0(\lambda) = 1,\quad P_1(\lambda) = c_1-\lambda\ ,\ Q_0(\lambda) = 0,$$
$$Q_1(\lambda) = 1\quad ,\ Q_2(\lambda) = c_2-\lambda,$$

then the initial values for the sequence (3.4.15) can be obtained from equation (3.4.10) and (3.4.11). The initial values of the sequence are,

$$\left.\begin{aligned}&R_0(\lambda) = 0,\ A_0 = 0,\ B_0 = 1,\ R_1(\lambda) = 0,\ A_1 = -1,\ B_1 = C_1\\ &S_0(\lambda) = 0,\ G_0 = 0,\ H_0 = 0,\ S_1(\lambda) = 0,\ G_1 = 0,\ H_1 = 1\end{aligned}\right\} \tag{3.4.16}$$

For any trial value of E and F the coefficients of the quadratic factor with starting values given in equation (3.4.16) the sequence given by equation (3.4.15) can be calculated, then from equation (3.4.13) the values of A_N and B_N are obtained. If A_N and B_N are zero then the quadratic factor $(\lambda^2-E\lambda-F)$ is a factor of $P_N(\lambda)$ and a solution has been found. So the problem is to find an E and F such

that the non-linear equations,

$$A_N(E,F) = B_N(E,F) = 0 \quad . \tag{3.4.17}$$

For arbitrary values of E and F the relationships (3.4.17) are not in general satisfied, so correction factors ΔE, ΔF must be found such that,

$$A_N(E+\Delta E,F+\Delta F) = B_N(E+\Delta E,F+\Delta F) = 0 \ . \tag{3.4.18}$$

This is achieved by dividing $R_i(\lambda)$, i=1,2,...,N and also S_i, i=1,2,...,N-1, by the same trial quadratic factor. This produces similar recurrence sequences to those obtained above and enables the corrections to the quadratic factors ΔE, ΔF to be calculated.

Synthetic division of the polynomials $R_i(\lambda)$, i=1,N and $S_i(\lambda)$, i=1,N-1 by the trial quadratic factor $(\lambda^2-E\lambda-F)$ yield,

$$\left.\begin{aligned} R_i(\lambda) &= (\lambda^2-E\lambda-F)T_i(\lambda) + L_i\lambda + M_i \ , \\ S_i(\lambda) &= (\lambda^2-E\lambda-F)U_i(\lambda) + V_i\lambda + W_i \ , \end{aligned}\right\} \quad i=1,2,.....,N-1 \tag{3.4.19}$$

$$R_N(\lambda) = (\lambda^2-E\lambda-F)T_N(\lambda) + L_N\lambda + M_N \ , \tag{3.4.20}$$

Substituting in equation (3.4.13) from equations (3.4.19) and (3.4.20) leaves,

$$\left.\begin{aligned} (\lambda^2-E\lambda-F)T_N(\lambda)+L_N\lambda+M_N = {} & (c_N-\lambda)((\lambda^2-E\lambda-F)T_{N-1}(\lambda)+L_{N-1}\lambda+M_{N-1}) \\ & -b_N^2((\lambda^2-E\lambda-F)T_{N-2}(\lambda)+L_{N-2}\lambda+M_{N-2}) \\ & -a.ad((\lambda^2-E\lambda-F)U_{N-1}+V_{N-1}\lambda+ W_{N-1}) \\ & -A_{N-1} \ . \end{aligned}\right\} \tag{3.4.21}$$

Equating the coefficients of $(\lambda^2-E\lambda-F)$, λ^1 and λ^0 in equation (3.4.21) gives,

$$\left.\begin{aligned} T_N(\lambda) &= (c_N-\lambda)T_{N-1}(\lambda)-L_{N-1}-b_N^2T_{N-2}-a.ad\ U_{N-1} \\ L_N &= (c_N-E)L_{N-1}-M_{N-1}-b_N^2L_{N-2}-a.ad\ V_{N-1} \\ M_N &= c_NM_{N-1}-FL_{N-1}-b_N^2M_{N-2}-a.ad\ W_{N-1}-A_{N-1} \ . \end{aligned}\right\} \tag{3.4.22}$$

Next the $R_i(\lambda)$ and $S_i(\lambda)$ in equation (3.4.15) are substituted from equations (3.4.19) to complete the division yielding,

$$\left.\begin{aligned}
(\lambda^2-E\lambda-F)T_i(\lambda)+L_i\lambda+M_i &= (c_i-\lambda)((\lambda^2-E\lambda-F)T_{i-1}(\lambda)+L_{i-1}\lambda+M_{i-1}) \\
&\quad -A_{i-1}-b_i^2((\lambda^2-\lambda E-F)T_{i-2}(\lambda)+L_{i-2}\lambda+M_{i-2}) \\
(\lambda^2-E\lambda-F)U_i(\lambda)+V_i\lambda+W_i &= (c_i-\lambda)((\lambda^2-E\lambda-F)S_{i-1}(\lambda)+V_{i-1}\lambda+W_{i-1}) \\
&\quad -G_{i-1}-b_i^2((\lambda^2-E\lambda-F)S_{i-2}(\lambda)+V_{i-2}\lambda+W_{i-2}).
\end{aligned}\right\} \quad i=1,\ldots,N-1 \qquad (3.4.23)$$

The coefficients of $(\lambda^2-E\lambda-F)$, λ^1, and λ^0, can now be equated in equation (3.4.23) and this will yield the following recursive relationships,

$$\left.\begin{aligned}
T_i(\lambda) &= (c_i-\lambda)T_{i-1}(\lambda)-L_{i-1}-b_i^2T_{i-2}(\lambda)\ , \\
L_i &= (c_i-J)L_{i-1}-b_i^2L_{i-2}-M_{i-1}\ , \\
M_i &= c_iM_{i-1}-KL_{i-1}-b_i^2M_{i-2}-A_{i-1}\ , \\
U_i(\lambda) &= (c_i-\lambda)U_{i-1}(\lambda)-V_{i-1}-b_i^2U_{i-2}\ , \\
V_i &= (c_i-J)V_{i-1}-b_i^2V_{i-2}-W_{i-1}\ , \\
W_i &= c_iW_{i-1}-KV_{i-1}-b_i^2W_{i-2}-G_{i-1}\ ,
\end{aligned}\right\} \quad i=1,2,\ldots,N-1, \qquad (3.4.24)$$

Now using the initial values as given in (3.4.16) the initial values for the sequences given in equations (3.4.24) can be calculated and are,

$$\begin{aligned}
&T_2 = 0,\ L_2 = 0,\ M_2 = 1,\ T_3 = 0,\ L_3 = -1,\ M_3 = C_3-A_3 \\
&U_2 = 0,\ V_2 = 0,\ W_2 = 0,\ U_3 = 0,\ V_3 = 0,\ U_3 = 0
\end{aligned} \qquad (3.4.25)$$

Thus for any trial quadratic factor with starting values as given by equations (3.4.25) the recursive sequence of equations (3.4.24) can be calculated. Then from equation (3.4.22) using the values just found M_N and L_N can be obtained. These two values and the values of A_N and B_N found earlier can now be used to calculate the two correction factors ΔE, and ΔF as follows,

$$\left.\begin{aligned}
\alpha &= F\,L_N + E\,(M_N + E\,L_N)\ , \\
\beta &= (M_N + E\,L_N)\,(M_N + E\,L_N) - \alpha L_N\ , \\
\Delta E &= (L_N(B_N + E\,A_N) - (M_N + E\,L_N)A_N)/\beta\ , \\
\Delta F &= (\alpha A_N - (M_N + E\,L_N)\,(B_N + E\,A_N))/\beta\ .
\end{aligned}\right\} \qquad (3.4.26)$$

Now after the corrections have been found new values for E and F are calculated

$$E = E + \Delta E, \qquad F = F + \Delta F \quad , \tag{3.4.27}$$

and these are used to form a new quadratic factor $(\lambda^2-(E\Delta+E)\lambda-(F+\Delta F))$. The whole process can now be repeated with new factors until A_N and B_N are zero, or a suitable stopping criterion is achieved. At this point the quadratic equation,

$$\lambda^2 - E\lambda - F = 0 \tag{3.4.28}$$

is solved to yield the required two eigenvalues. The sequences of polynomials $R_i(\lambda)$, i=1,2,...,N, and $S_i(\lambda)$, i=1,2,...,N-1 of maximum degree N-2, N-3, respectively defined by equation (3.4.15) and (3.4.13) can now be used to determine further pairs of eigenvalues in the same manner. It should be noted that as N has been effectively reduced by two at each stage the recursive sequences are shorter and further eigenvalues are progressively quicker and easier to calculate. The convergence of the algorithm is [illegible] quadratic when close to a pair of eigenvalues, (Wilkinson, 1965). However in practical experiment the algorithm often took a large number of iterations before it "settled" on a pair of eigenvalues, and then converged rapidly. A lot of effort was put into trying to find some way of getting a close guess to a pair of eigenvalues, thus cutting out the initial 'hunting' for a pair to converge to. This failed, mainly because even if quite close estimates were obtained for a pair of eigenvalues, the resulting quadratic factors need not be close to the actual ones. As an example, if the eigenvalues are 0.1 and 1000 then the quadratic factor is $\lambda^2-100.1\lambda + 100$. If estimates are found of -0.1 and 1000 then the quadratic factor is $\lambda^2- 999.9\lambda - 100$, and convergence will be to a different pair of eigenvalues. As there are N eigenvalues they can be combined to produce N(N-1)/2 quadratic

functions. The fact that there are a large number of factors to chose from does not increase the probability of finding one, but merely serves to slow the convergence process down. For this reason arbitrary starting values from the diagonal elements are chosen.

The main value of this algorithm is that it can find complex eigenvalues without having to work in complex arithmetic which is time consuming on a computer. Also as all correction factors and remainders can be calculated without explicitly calculating the new polynomials produced at each stage ($R_i(\lambda)$, $T_i(\lambda)$, i=1,N and $S_i(\lambda)$, $U_i(\lambda)$, i=N-1) no storage need be reserved for them in the computer and thus storage is kept to a minimum.

3.5 RESULTS

The algorithm to perform the Bairstow method as described in (3.4) is given in Appendix 1 in program 3. A number of tests were then used to demonstrate the performance of this algorithm.

First the algorithm was used to find the eigenvalues of a (14×14) matrix of the same form as that given in (3.3.1). These experimental results were then compared with the theoretical results and found to be accurate to 9 significant figures as in Table 3.5.1. The program took 1.5 seconds to obtain the results.

λ_i	Program 2 Result	Theoretical Result
1	$1.565437219 \times 10^{-11}$	$2.168404325 \times 10^{-19}$
2	0.1980622644	0.1980622639
3	0.1980622637	0.1980622642
4	0.7530203958	0.7530203959
5	0.7530203961	0.7530203963
6	1.554958131	1.554958132
7	1.554958134	1.554958132
8	2.445041865	2.445041867
9	2.445041870	2.445041868
10	3.242697959	3.242697960
11	3.242697964	3.242697960
12	3.801937739	3.801937736
13	3.801937739	3.801937736
14	4.000000000	4.000000000

TABLE 3.5.1

Next, the algorithm was tried on a (20×20) unsymmetric matrix of the form,

$$\begin{bmatrix} -1 & 10 & & & & & & & & & & & & & & & & & & -10 \\ 10 & 1 & -10 & & & & & & & & & & & & & & & & & \\ & -10 & -1 & 10 & & & & & & & & & & & & & & & \\ & & 10 & \ddots & \ddots & & & & & & & & & & & & & & \\ & & & \ddots & \ddots & \ddots & & & & & & & & & & & & & \\ & & & & \ddots & \ddots & \ddots & & & & & & & & & & & & \\ & & & & & \ddots & \ddots & \ddots & & & & & & & & & & & \\ & & & & & & \ddots & \ddots & \ddots & & & & & & & & & & \\ & & & & & & & \ddots & \ddots & \ddots & & & & & & & & & \\ & & & & & & & & \ddots & \ddots & \ddots & & & & & & & & \\ & & & & & & & & & \ddots & \ddots & \ddots & & & & & & & \\ & & & & & & & & & & \ddots & \ddots & \ddots & & & & & & \\ & & & & & & & & & & & \ddots & \ddots & \ddots & & & & & \\ & & & & & & & & & & & & \ddots & \ddots & \ddots & & & & \\ & & & & & & & & & & & & & \ddots & \ddots & \ddots & & & \\ & & & & & & & & & & & & & & \ddots & \ddots & \ddots & & \\ & & & & & & & & & & & & & & & \ddots & \ddots & \ddots & \\ & & & & & & & & & & & & & & & & \ddots & \ddots & -10 \\ 10 & & & & & & & & & & & & & & & & & -10 & 1 \end{bmatrix} \qquad (3.5.1)$$

This matrix is also centro-antisymmetric which means the eigenvalues occure in pairs to the same modulus but of different signs. The results are compared with those obtained from a N.A.G. routine using a QR process in Table 3.5.2.

λ_j	Program 2 Results	N.A.G. Results
1	-19.82166982	-19.82166983
2	19.82166983	19.82166983
3	-19.04739875	-19.04739875
4	19.04739873	19.04739874
5	-18.22010282	-18.22010282
6	18.22010284	18.22010282
7	-16.21121213	-16.21121213
8	16.21121212	16.21121216
9	-15.14803511	-15.14803512
10	15.14803511	15.14803511
11	-11.79816092	-11.79816093
12	11.79816092	11.79816092
13	-10.85895931	-10.85895931
14	10.85895931	10.85895932
15	-6.260718898	-6.260718904
16	6.260718898	6.260718899
17	-5.722699838	-5.722699833
18	5.722699838	5.722699838
19	-1.000000000	-0.999999994
20	1.000000000	1.000000002

TABLE 3.5.2

The results were obtained in 7 seconds by program 3, and 3 seconds by the N.A.G. routine. These results, whilst not conclusive, indicate that an accuracy of at least 9 significant figures has been obtained.

As a final example the program was tested on a sparse unsymmetric matrix with corner elements of order 14. The results below were accurate to at least 9 significant figures, and were obtained in 4 seconds.

The matrix used was of the form

$$
\begin{bmatrix}
1 & 9 & & & & & & & & & & & 20 \\
9 & 2 & 9 & & & & & & & & & & \\
 & 9 & 3 & 9 & & & & & & & & & \\
 & & 9 & 4 & 9 & & & & & & & & \\
 & & & 9 & 5 & 9 & & & & & & & \\
 & & & & 9 & 6 & 9 & & & & & & \\
 & & & & & 9 & 7 & 9 & & & & & \\
 & & & & & & 9 & 8 & 9 & & & & \\
 & & & & & & & 9 & 9 & 9 & & & \\
 & & & & & & & & 9 & 10 & 9 & & \\
 & & & & & & & & & 9 & 11 & 9 & \\
 & & & & & & & & & & 9 & 12 & 9 \\
 & & & & & & & & & & & 9 & 13 & 9 \\
1 & & & & & & & & & & & & 9 & 14
\end{bmatrix}
\qquad (3.5.2)
$$

The eigenvalues of this matrix are

	Real	Imaginary
1	-13.274701666	0.00000000000
2	-9.6902838613	0.00000000000
3	-7.1426890350	0.00000000000
4	-3.0104685765	-0.6042324007
5	-3.0104685765	0.6042324007
6	3.7326625824	-1.293792672
7	3.7326625824	1.293792672
8	11.267337419	-1.293792668
9	11.267337419	1.293792668
10	18.010468575	-0.6042323963
11	18.010468575	0.6042323963
12	22.142689034	0.00000000000
13	24.690283861	0.00000000000
14	28.274701666	0.00000000000

TABLE 3.5.2

3.6 DETERMINATION OF THE EIGENVALUES OF SYMMETRIC AND UNSYMMETRIC MATRICES BY NEWTON'S METHOD

Other methods for finding the eigenvalues of unsymmetric matrices of the same form as (3.4.1) have been considered, and here Newton's method is quite effective.

The method is similar to that described by Evans (1971). First the sequences (3.4.2), (3.4.3), and (3.4.4) are differentiated with respect to λ to produce,

$$\left.\begin{aligned} &\frac{d}{d\lambda} P_0(\lambda) = P_0'(\lambda) = 0 \quad , \\ &\frac{d}{d\lambda} P_1(\lambda) = P_1'(\lambda) = -1 \quad , \\ &\frac{d}{d\lambda} P_i(\lambda) = P_i'(\lambda) = (c_i-\lambda)P_{i-1}'(\lambda) - P_{i-1}(\lambda) - b_i d_i P_{i-2}'(\lambda), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i=2,3....,N-1, \end{aligned}\right\} \quad (3.6.1)$$

and,

$$\left.\begin{aligned} &\frac{d}{d\lambda} Q_1(\lambda) = Q_1'(\lambda) = 0 \quad , \\ &\frac{d}{d\lambda} Q_2(\lambda) = Q_2'(\lambda) = -1 \quad , \\ &\frac{d}{d\lambda} Q_i(\lambda) = Q_i'(\lambda) = (c_i-\lambda)Q_{i-1}'(\lambda) - Q_{i-1}(\lambda) - b_i d_i Q_{i-2}'(\lambda), \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i=3,4,...,N-1. \end{aligned}\right\} \quad (3.6.2)$$

with finally,

$$P_N'(\lambda) = (c_N-\lambda)P_{N-1}'(\lambda) - P_{N-1}(\lambda) - b_N d_N P_{N-2}'(\lambda) - b_1 d_1 Q_{N-1}'(\lambda). \quad (3.6.3)$$

Now (3.4.2) and (3.6.3) can be used to find the eigenvalues of matrix (3.4.1) in a Newton's iterative method of the form,

$$\lambda^{(k+1)} = \lambda^{(k)} - P_N(\lambda^{(k)})/P_N'(\lambda^{(k)}), \quad k \geq 0 \quad , \quad (3.6.4)$$

where $\lambda^{(0)}$ is an initial estimate.

When one or more of the eigenvalues has been computed, then in order to avoid re-determining those eigenvalues already found a technique known as dividing out the root is needed to suppress the known eigenvalues, thus instead of iterating with $P_N(\lambda)$, $G_N(\lambda)$ is now used where,

$$G_N(\lambda) = P_N(\lambda) / \prod_{i=1}^{T} (\lambda-\lambda_i) \quad , \tag{3.6.5}$$

and λ_i, i=1,2,...,T are the T eigenvalues already found. Now $G_N(\lambda)$ can be differentiated with respect to λ giving,

$$\frac{d}{d\lambda} G_N(\lambda) = G'_N(\lambda) = G_N(\lambda) \left(\frac{P'_N(\lambda)}{P_N(\lambda)} - \sum_{i=1}^{T} (\lambda-\lambda^{(i)})^{-1}\right) , \tag{3.6.6}$$

Newton's iteration is now of the form,

$$\lambda^{(k+1)} = \lambda^{(k)} - G_N(\lambda^{(k)})/G'_N(\lambda^{(k)}), \quad k\geq 0 \quad , \tag{3.6.7}$$

and substituting for $G_N(\lambda)/G'_N(\lambda)$ using equation (3.6.5) the Newton iteration becomes,

$$\lambda^{(k+1)} = \lambda^{(k)} - 1 \Big/ \left(\frac{P'_N(\lambda^{(k)})}{P_N(\lambda^{(k)})} - \sum_{i=1}^{T} (\lambda^{(k)}-\lambda_i)^{-1}\right) \quad , \tag{3.6.8}$$

The relationship (3.6.8) can now be used to determine all the eigenvalues of the matrix.

This method was programmed on the I.C.L. 1904S in complex arithmetic in ALGOL 60. The eigenvalues obtained were all accurate to machine accuracy. The method however had two drawbacks, which detract from the usefulness of the method. Firstly for every test matrix of size larger than 30×30 the values of $P_i(\lambda)$, $P'_i(\lambda)$, i=1,2,...,N oscillated wildly, and at some point overflowed the computer word length. The problem here is that in the vicinity of an eigenvalue the values of $P_N(\lambda)$, $P'_N(\lambda)$ are 'well behaved' and do not overflow the computer word length. However when an estimate at an eigenvalue is made, after just finding the previous one, there is no guarantee that this will be close to any remaining eigenvalue and it is at this point that oscillation can, and usually does, set in. The other drawback is that compared to the available N.A.G. routines this method is a lot slower, and takes from 3 to 4 times longer. As this method is specialised to a particular type of matrix the storage requirement in computer memory is O(2N) (to allow for complex elements) words, whereas

the storage requirement for the more generalised N.A.G. routine is $O(4N^2)$ words.

To overcome the first problem, that of overflow in the computer, the sequence (3.2.16) can be used instead of sequence (3.4.2). For the sake of clarity the sequence for a symmetric matrix is considered, the extension to the unsymmetric case is fairly simple.

First the sequence (3.2.16) is differentiated with respect to λ, to give,

$$\left.\begin{aligned}
\frac{ds_0(\lambda)}{d\lambda} &= s_0'(\lambda) = 0 \quad , \\
\frac{ds_1(\lambda)}{d\lambda} &= s_1'(\lambda) = -1 \quad , \\
\frac{ds_i(\lambda)}{d\lambda} &= s_i'(\lambda) = -1 + b_i^2\, s_{i-1}'(\lambda)/(s_{i-1}(\lambda))^2, \quad i=2,3,\ldots,N-1
\end{aligned}\right\}$$

and

$$\left.\begin{aligned}
\frac{dr_1}{d\lambda} &= r_1' = 0 \\
\frac{dr_i}{d\lambda} &= r_i' = -b_i(s_{i-1}(\lambda)r_{i-1}'-s_{i-1}'(\lambda)r_{i-1})/(s_{i-1}(\lambda))^2, \\
&\qquad\qquad i=2,3,\ldots,N-1
\end{aligned}\right\} \tag{3.6.9}$$

with

$$\left.\begin{aligned}
\frac{ds_N(\lambda)}{d\lambda} &= s_N'(\lambda) = -1 - \sum_{j=1}^{N-1}(2s_j(\lambda)r_j r_j' - r_j^2 s_j'(\lambda))/(s_j(\lambda))^2 \\
&\quad + 2b_N(s_{N-1}(\lambda)r_{N-1}' - s_{N-1}'(\lambda)r_{N-1})/(s_{N-1}(\lambda))^2 \\
&\quad + b_N^2\, s_{N-1}'(\lambda)/(s_{N-1}(\lambda))^2 \quad .
\end{aligned}\right\}$$

From equation (3.2.23) it is seen that,

$$P_i(\lambda) = \prod_{j=1}^{i} s_j(\lambda) \quad , \tag{3.6.10}$$

and upon differentiating with respect to λ, (3.6.10) becomes,

$$\frac{dP_i(\lambda)}{d\lambda} = P_i'(\lambda) = \sum_{k=1}^{i}\Big(\prod_{\substack{j=1\\ j\neq k}}^{i} s_j(\lambda)\; s_k'(\lambda)\Big) \quad . \tag{3.6.11}$$

Now from (3.6.10) and (3.6.11) we have,

$$\frac{P_i'(\lambda)}{P_i(\lambda)} = \sum_{k=1}^{i} \Big(\prod_{\substack{j=1 \\ j\neq k}}^{i} s_j(\lambda)\, s_k'(\lambda) \Big) \Big/ \prod_{j=1}^{i} s_j(\lambda) , \tag{3.6.12}$$

$$= \sum_{j=1}^{i} \frac{s_j'(\lambda)}{s_j(\lambda)} . \tag{3.6.13}$$

Equation (3.6.13) can be substituted into equation (3.6.8) to give,

$$\lambda^{k+1} = \lambda^k - 1/\Big(\sum_{i=1}^{N} s_i'(\lambda)/s_i(\lambda) - \sum_{j=1}^{T} (\lambda^k - \lambda_j)^{-1} \Big), \tag{3.6.14}$$

This relationship is used instead of (3.6.8) and the eigenvalues of any order matrix can be found without fear of overflow occurring. The work involved in calculating the new sequence and Newton's formula is similar to that of the previous method.

The program to perform the above algorithm is included in Appendix 1 in program 4.

The sequence for an unsymmetric matrix of the form (3.4.1) is obtained by similar methods to those already described and is given below,

$$\left.\begin{aligned} &s_0(\lambda) = c_1 - \lambda , \\ &s_i(\lambda) = c_i - \lambda - b_i d_i / s_{i-1}(\lambda), \quad i=2,3,\ldots,N-1, \\ &\text{and} \\ &R_1 = d_1 \qquad\qquad P_1 = b, \\ &R_i = -R_{i-1} d_i / s_{i-1}(\lambda), \quad P_i = P_{i-1} b_i / s_{i-1}(\lambda), \quad i=2,3,\ldots,N-1 \\ &\text{with finally,} \\ &s_N(\lambda) = c_N - \sum_{i=1}^{N-1} P_i R_i / s_i(\lambda) - d_N R_{N-1} / s_{N-1}(\lambda) - b_N P_{N-1} / s_{N-1}(\lambda) \\ &\qquad\qquad - b_N d_N / s_{N-1}(\lambda) \end{aligned}\right\} \tag{3.6.15}$$

This can now be differentiated with respect to λ to give the following equations

$$
\left.
\begin{aligned}
s'_1(\lambda) &= -1 , \\
s'_i(\lambda) &= -1-b_i d_i s'_{i-1}(\lambda)/s^2_{i-1}(\lambda), \quad i=1,2,\ldots,N-1, \\
\text{and} \qquad R'_1 &= 0 \\
R'_i &= d_i(R_{i-1}s'_{i-1}(\lambda)-s_{i-1}(\lambda)R'_{i-1})/s^2_{i-1}(\lambda), \quad i=1,2,\ldots,N-1, \\
P'_1 &= 0 \\
P'_i &= b_i(P_{i-1}s'_{i-1}(\lambda)-s_{i-1}(\lambda)P'_{i-1})/s^2_{i-1}(\lambda) , \quad i=1,2,\ldots,N-1, \\
\text{with finally,} & \\
s'_N &= -1 - \sum_{i=1}^{N-1} (s_i(\lambda)(P'_i R_i + R'_i P_i) - s'_i(\lambda)P_i R_i)/s^2_i(\lambda) \\
&\quad + d_N(R_{N-1}s'_{N-1}(\lambda)-R'_{N-1}s_{N-1}(\lambda))/s^2_{N-1}(\lambda) \\
&\quad + b_N(s'_{N-1}(\lambda)P_{N-1}-s_{N-1}(\lambda)P'_{N-1})/s^2_{N-1}(\lambda) \\
&\quad + b_N d_N s'_{N-1}(\lambda)/s^2_{N-1}(\lambda)
\end{aligned}
\right\} \quad (3.6.16)
$$

The sequences of $s_i(\lambda)$, $s'_i(\lambda)$, i=1,N obtained in (3.6.15) and (3.6.16) can now be used in equations (3.6.11)-(3.6.14) to obtain the eigenvalues of the unsymmetric matrix of (3.4.1). The program to perform this algorithm is given in Appendix 1 in program 5. It is written in ALGOL 68R to take advantage of the superior facilities for handling complex arithmetic in this language.

3.7 RESULTS

Program 4 was test run on the 11×11 matrix (3.3.1), the results were identical to those given in Table 3.3.1, and the computer took 2 seconds to perform the calculation. Comparison beyond this is not really meaningful, as with program 4 the eigenvalues are obtained in no particular set order and the most useful feature of the bisection algorithm is its ability to pick out selected eigenvalues. In this example all but one of the eigenvalues are double and as a result

convergence in this case was linear (Hildebrand (1974)). The type of problem where it is preferable to use Newton's method is when all the eigenvalues are required and none, or at least very few, multiple roots are expected.

Program 5 was test run on a number of matrices, and these results are compared in the tables against results obtained from N.A.G. routines. None of the N.A.G. routines were exactly tailored to fit the particular problem but it was thought that the routine F02 AJA was most efficient. This routine reduces the matrix to upper Hessenberg form using stabilised similarity transformations, then uses a modified LR algorithm to obtain the eigenvalues. To test the routine the matrix (3.3.1) was first tried, and again the table 3.3.1 was duplicated. As the program was performing complex arithmetic it took 21 seconds to obtain the answers.

Next an unsymmetric matrix, order 20, of the following form was tested on the program,

$$
\begin{bmatrix}
1,5 & & & & & & & & & & & & & & & & & & & -3 \\
2,2,6 & & & & & & & & & & & & & & & & & & & \\
 & 4,3,7 & & & & & & & & & & & & & & & & & & \\
 & & 2,4,9 & & & & & & & & & & & & & & & & & \\
 & & & 3,5,3 & & & & & & & & & & & & & & & & \\
 & & & & 5,6,-2 & & & & & & & & & & & & & & & \\
 & & & & & 6,7,-4 & & & & & & & & & & & & & & \\
 & & & & & & 8,8,5 & & & & & & & & & & & & & \\
 & & & & & & & 9,9,3 & & & & & & & & & & & & \\
 & & & & & & & & 1,10,6 & & & & & & & & & & & \\
 & & & & & & & & & 2,11,0 & & & & & & & & & & \\
 & & & & & & & & & & 3,12,6 & & & & & & & & & \\
 & & & & & & & & & & & 0,13,5 & & & & & & & & \\
 & & & & & & & & & & & & 5,14,52 & & & & & & & \\
 & & & & & & & & & & & & & 4,15,-12 & & & & & & \\
 & & & & & & & & & & & & & & 2,-1,2 & & & & & \\
 & & & & & & & & & & & & & & & 5,-7,2 & & & & \\
 & & & & & & & & & & & & & & & & 3,-5,5 & & & \\
 & & & & & & & & & & & & & & & & & 5,-100,2 & & \\
2 & & & & & & & & & & & & & & & & & & 1,-2 &
\end{bmatrix}
\qquad (3.7.1)
$$

The results are compared with those obtained by using the N,A,G, routine, F02 AJA,

Program 5		N,A,G, Routine	
Real	Imaginary	Real	Imaginary
-100.2828912	0.000000000	-100.2828912	0.000000000
-9.315123512	0.000000000	-9.315123513	0.000000000
-4.418121111	0.000000000	-4.418121109	0.000000000
-3.526424826	0.000000000	-3.526424826	0.000000000
-1.340555599	-1.379912314	-1.340555600	-1.379912312
-1.340555599	1.379912314	-1.340555599	1.379912312
-0.1133422775	-2.756343922	-0.1133422791	-2.756343920
0.6045356208	0.000000000	0.6045356209	0.000000000
3.433824914	0.000000000	3.433824913	0.000000000
6.439112820	-4.516691649	6.439112820	-4.516691649
6.439112820	4.516691649	6.439112820	4.516691648
6.639222683	0.000000000	6.639222684	0.000000000
7.934262224	0.000000000	7.934262224	0.000000000
11.56851719	0.000000000	11.56851719	0.000000000
12.00000000	0.000000000	12.00000000	0.000000000
13.02409163	0.000000000	13.02409163	0.000000000
13.10939899	0.000000000	13.10939899	0.000000000
14.95366508	0.000000000	14.95366508	0.000000000
29.30461241	0.000000000	29.30461241	0.000000000

TABLE 3.7.1

Program 5 took 32 seconds to obtain the answers whereas the N,A.G, routine obtained the answers in 9 seconds, This speed up is always obtained by transformation methods compared to a Sturm sequence method, when the matrix can be stored in core. However as the storage for program 5 was O(N) words and for the N,A,G, routine $O(N^2)$ words then it can be seen that the N,A,G, routine will soon run into storage difficulties on large matrices, program 5 is then the preferred method to use. This

method is in general faster than other methods of dealing with this type of matrix e.g. (Golub 1973). The accuracy of the results can be determined, and in practice is seen to be accurate on even larger matrices (Chapter 5), and the order of matrix that can be readily solved approaches the amount of core store available on the computer, the only limiting factor being the time available. For large systems where the core requirments of the NAG Routines become excessive, using equations (3.6.15)+(3.6.16) results were easily obtained for N=200 to 1000.

CHAPTER 4

NEW STRATEGIES FOR DERIVING THE EIGENVALUES AND EIGENVECTORS OF CENTRO-SYMMETRIC MATRICES

4.1 INTRODUCTION

In this chapter a new strategy is proposed for deriving the numerical solution of the matrix eigenvalue equation,

$$A\underline{x} = \lambda\underline{x} \,, \qquad (4.1.1)$$

i.e. the determination of the eigenvalue and eigenvector $\underline{x}$ of the N×N centro-symmetric tridiagonal matrix A as given in (4.2.1).

The symmetric Gaussian elimination or folding algorithm as outlined in Evans and Hatzopoulos (1975) is based on the strategy of performing a Gaussian elimination process at the top left hand and bottom right hand corner of the matrix $(A-\lambda I)$ concurrently. By noting that the coefficients in the top left hand corner are identical to the elements in the bottom right hand corner in reverse order when the matrix A is centro-symmetric, a reduced and compact form of the Sturm sequence can be obtained and used to isolate the eigenvalues in a bisection process.

A similar technique can also be used to derive the eigenvectors by an inverse iteration process where a Gaussian elimination process is carried out in which a pivoting strategy has been incorporated to maintain numerical stability.

Similar strategies to those outlined above can now be used to solve equation (4.1.1) using two processors when the matrix A has a more general form i.e. A is symmetric and tridiagonal. Even though the upper left hand, and bottom right hand corner elements are not the same each processor can commence the Gaussian Elimination process in opposite corners, and work towards the centre.

The Gaussian elimination is therefore speeded up by using the two processors. The algorithms were then run on a parallel processing machine with two processors available and results, and the outline of an improved algorithm given.

4.2 DERIVATION OF THE STURM SEQUENCE FOR A CENTRO-SYMMETRIC MATRIX

Let A be a centro-symmetric (N×N) matrix such that

$$A \equiv \begin{bmatrix} c_1 & b_2 & & & & & \\ b_2 & c_2 & b_3 & & & 0 & \\ & b_3 & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & b_3 & \\ & 0 & & & b_3 & c_2 & b_2 \\ & & & & & b_2 & c_1 \end{bmatrix} . \qquad (4.2.1)$$

If each of the sub-diagonal elements b_i, i=2,3,...(N/2)....2 of A-λI, is now eliminated successively using a Gauss reduction process without pivoting, it can easily be shown that the matrix (A-λI) becomes

$$\begin{bmatrix} Q_1(\lambda) & b_2 & & & & \\ & Q_2(\lambda) & \ddots & & 0 & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & 0 & & & \ddots & b_N \\ & & & & & Q_N(\lambda) \end{bmatrix} , \qquad (4.2.2)$$

where $Q_1(\lambda) = c_1-\lambda$, (4.2.3)

and if N is even,

$$\left.\begin{aligned} Q_i(\lambda) &= c_i-\lambda-b_i^2/Q_{i-1}(\lambda), \quad i=2,3,\ldots,N/2, \\ Q_i(\lambda) &= c_{N+1-i}-\lambda-b_{N+2-i}^2/Q_{i-1}(\lambda), \quad i=N/2+1,\ldots,N, \end{aligned}\right\} \qquad (4.2.4)$$

or if N is odd,

$$\left.\begin{aligned} Q_i(\lambda) &= c_i-\lambda-b_i^2/Q_{i-1}(\lambda), \quad i=2,3,\ldots,(N+1)/2, \\ Q_i(\lambda) &= c_{N+1-i}-\lambda-b_{N+2-i}^2/Q_{i-1}(\lambda), \quad i=2,3,\ldots,(N+3)/2, \end{aligned}\right\} \qquad (4.2.5)$$

Then $P_i(\lambda)$, (i=0,N, $P_0(\lambda)=1$) the leading principal minors of the matrix (A-λI) are given by the recursive relationship,

$$\left.\begin{aligned} P_0(\lambda) &= 1 \quad , \\ P_1(\lambda) &= c_1-\lambda \quad , \\ P_2(\lambda) &= (c_2-\lambda)P_1(\lambda)-b_2^2P_0(\lambda) \quad , \\ &\vdots \\ P_i(\lambda) &= (c_i-\lambda)P_{i-1}(\lambda)-b_i^2P_{i-2}(\lambda), \quad i=3,\ldots,N, \end{aligned}\right\} \tag{4.2.6}$$

it can easily be seen from equations (4.2.6), (4.2.5), (4.2.4) and (4.2.3) that,

$$P_i(\lambda) = Q_1(\lambda)Q_2(\lambda).....Q_i(\lambda) \quad , \tag{4.2.7}$$

and therefore that,

$$\frac{P_i(\lambda)}{P_{i-1}(\lambda)} = Q_i(\lambda) \quad . \tag{4.2.8}$$

The $P_i(\lambda)$, i=1,....,N now form a Sturm sequence for the matrix (4.2.1) and as in the previous chapter the signs of the $Q_i(\lambda)$ can now be used in a bisection process to isolate the required eigenvalues as used by Barth et al (1967), and Strang and Fix (1973).

Now since the matrix A is centro-symmetric, the simple stragegy of eliminating from both ends of the diagonal of the matrix $(A-\lambda I)$ simultaneously can easily be shown to yield the following reduced form,

$$\begin{bmatrix} q_1(\lambda) & b_2 & & & & & & & \\ 0 & q_2(\lambda) & b_3 & & & & & & \\ & 0 & q_3(\lambda) & & & & & & \\ & & 0 & \ddots & & & & & \\ & & & 0 & B & 0 & & & \\ & & & & 0 & \ddots & q_3(\lambda) & 0 & \\ & & & & & & b_3 & q_2(\lambda) & 0 \\ & & & & & & & b_2 & q_1(\lambda) \end{bmatrix} \quad . \tag{4.2.9}$$

If N is odd, the submatrix B is given by,

$$B = \begin{bmatrix} q_{\frac{N+1}{2}}(\lambda) & b_{\frac{N+3}{2}} & 0 \\ 0 & q_{\frac{N+3}{2}}(\lambda) & 0 \\ 0 & b_{\frac{N+3}{2}} & q_{\frac{N+1}{2}}(\lambda) \end{bmatrix} , \tag{4.2.10}$$

and if N is even, the submatrix B is given by,

$$B = \begin{bmatrix} q_{\frac{N}{2}}(\lambda) & b_{\frac{N+2}{2}} & 0 \\ 0 & q_{\frac{N+2}{2}}(\lambda) & 0 \\ 0 & b_{\frac{N-2}{2}} & q_{\frac{N-2}{2}}(\lambda) \end{bmatrix} . \tag{4.2.11}$$

If N is odd let,

$$s = \frac{N-1}{2} , \tag{4.2.12}$$

and if N is even let,

$$s = \frac{N+2}{2} . \tag{4.2.13}$$

The recursive relationship for the $q_i(\lambda)$, i=1,2,...,s can now be written, for N odd,

$$\left.\begin{aligned} q_1(\lambda) &= c_1-\lambda , \\ q_i(\lambda) &= c_i-\lambda-b_i^2/q_{i-1}(\lambda) , \quad i=1,2,\ldots,s-1, \\ q_s(\lambda) &= c_s-\lambda-2b_s^2/q_{s-1}(\lambda), \end{aligned}\right\} \tag{4.2.14}$$

for N even,

$$\left.\begin{aligned} q_1(\lambda) &= c_1-\lambda , \\ q_i(\lambda) &= c_i-\lambda-b_i^2/q_{i-1}(\lambda), \quad i=2,3,\ldots,s-1, \\ q_s(\lambda) &= q_{s-1}(\lambda)-b_s^2/q_{s-1}(\lambda) , \end{aligned}\right\} \tag{4.2.15}$$

Thus a unique polynomial sequence of $q_i(\lambda)$ i=1,2,...,s have been developed for the given centro-symmetric matrix, which are the pivots in a Gaussian elimination process without pivoting. The $q_i(\lambda)$, i=1,s, can now be obtained from equations (4.2.14) or (4.2.15) without explicitly performing the elimination.

The half sequences given in (4,2,14) and (4,2,15) are no longer the usual Sturm sequences for the matrix (4,2,1), so the signs of the $q_i(\lambda)$ i=1,N must be shown to have the same number of negative signs as the $Q_i(\lambda)$ i=1,N.

This is easily done for both odd and even cases, the even case will only be considered here, the odd case follows similarly.

If we denote s=(N+2)/2, the matrix A in 4.2.1for N even can be shown thus:-

$$A = \begin{bmatrix} c_1 & b_2 & & & & & & & & & \\ b_2 & c_2 & b_3 & & & & & & & & \\ & b_3 & c_3 & \ddots & & & & & & & \\ & & \ddots & \ddots & b_{s-1} & & & & & & \\ & & & b_{s-1} & c_{s-1} & b_s & & & & & \\ & & & & b_s & c_{s-1} & b_{s-1} & & & & \\ & & & & & b_{s-1} & \ddots & \ddots & & & \\ & & & & & & \ddots & c_3 & b_3 & & \\ & & & & & & & b_3 & c_2 & b_2 & \\ & & & & & & & & b_2 & c_1 & \end{bmatrix} \qquad (4.2.16)$$

If now the columns and rows of A are interchanged as follows:- column s with column N, column s+1 with column N-1, column s+2 with column N-2, and on until all the columns (except the centre column in the case when (N-s) is even) have been interchanged. The rows corresponding to the column changes are now interchanged, and the matrix A becomes,

$$\bar{A} = \begin{bmatrix} \bar{c}_1 & b_2 & & & & & & \\ b_2 & c_2 & \ddots & & & & 0 & \\ & \ddots & \ddots & b_{s-1} & & & & \\ & & b_{s-1} & c_{s-1} & 0 & & & b_s \\ & & & 0 & c_1 & b_2 & & \\ & & 0 & & b_2 & \ddots & & \\ & & & & & \ddots & c_{s-1} & b_{s-1} \\ & & & b_s & & & b_{s-1} & c_s \end{bmatrix} \tag{4.2.17}$$

Now Gaussian elimination can be performed for the first s-1 rows of $\bar{A}$, which becomes,

$$\begin{bmatrix} \bar{R}_1(\lambda) & b_2 & & & & & & \\ 0 & R_2(\lambda) & b_3 & & & & 0 & \\ & 0 & \ddots & \ddots & & & & \\ & & \ddots & \ddots & b_{s-1} & & & \\ & & & & R_{s-1}(\lambda) & 0 & & b_s \\ & & & & 0 & c_1 & b_2 & \\ & & 0 & & & b_2 & \ddots & \\ & & & & & & \ddots & b_{s-1} \\ & & & & b_s & & b_{s-1} & c_{s-1} \end{bmatrix} \tag{4.2.18}$$

where
$$\left.\begin{aligned} R_1(\lambda) &= c_1 \quad , \\ R_i(\lambda) &= c_i - b_i^2/R_{i-1}(\lambda), \qquad i=2,3,\ldots,s-1 \end{aligned}\right\} \tag{4.2.19}$$

Performing one more step of elimination and (4.2.18) becomes,

$$\begin{bmatrix} R_1(\lambda) & b_2 & & & & & \\ 0 & R_2(\lambda) & b_3 & & & & \\ & 0 & \ddots & \ddots & & 0 & \\ & & & b_s & & & \\ & & & R_{s-1}(\lambda) & 0 & & b_s \\ & & & 0 & R_s(\lambda) & b_2 & \\ & 0 & & & b_2 & c_2 & \ddots \\ & & & & & \ddots & b_{s-1} \\ & & & & & b_{s-1} & c_{s-1}-b_s^2/R_s \end{bmatrix} \tag{4.2.20}$$

where $R_s(\lambda) = c_1$.

If the Gaussian elimination process is now pursued for the remaining s-2 rows, the matrix becomes

$$\begin{bmatrix} R_1(\lambda) & b_2 & & & & \\ & \ddots & \ddots & & 0 & \\ & & 0 & & & b_s \\ & & & R_s(\lambda) & b_2 & \\ & 0 & & & \ddots & b_{s-1} \\ & & & & & R_N(\lambda) \end{bmatrix} \tag{4.2.21}$$

where
$$\left.\begin{aligned} R_i(\lambda) &= c_{i-s+1} - b_{i-s+1}^2/R_{i-1}(\lambda), \quad i=s\ldots N-1 \\ R_N(\lambda) &= c_{s-1} - b_s^2/R_{s-1}(\lambda) - b_{s-1}^2/R_{N-1}(\lambda) \ . \end{aligned}\right\} \tag{4.2.22}$$

Then, from (4.2.19) and (4.2.22) it can be seen that,

$$R_i(\lambda) = R_{s+i-1}(\lambda) \ , \qquad i=1,2,\ldots,s-2, \tag{4.2.23}$$

and therefore,

$$R_N(\lambda) = R_{s-1}(\lambda) - b_2^2/R_{s-1}(\lambda) \tag{4.2.24}$$

The $R_i(\lambda)$, i=1,N are now the equivalent Sturm sequence for the matrix A, and the $R_i(\lambda)$ are identical to the $q_i(\lambda)$. Therefore the $q_i(\lambda)$ can be used in a bisection process to isolate the eigenvalues

of $\bar{A}$. However matrix $\bar{A}$ was obtained from matrix A by a series of similarity transforms, The two matrices are therefore similar and have the same eigenvalues.

Therefore, the equations (4.2.14) and (4.2.15) can be used in a bisection process to determine the eigenvalues of the matrix A in place of equations (4.2.4) and (4.2.5), provided it is noted that the $q_i(\lambda)$, i=1,s-1 (i=1,s-2 of N is even) occur twice on the main diagonal of (4.2.9).

The ALGOL 60 program to carry out the above procedure is given in Appendix 1 in program 6.

4.3 THE CALCULATION OF THE EIGENVECTORS OF A TRIDIAGONAL CENTRO-SYMMETRIC MATRIX BY INVERSE ITERATION

Suppose $\underline{V}$ is taken as the initial trial vector, then to find the eigenvector corresponding to given values of λ_j (say) using two steps of inverse iteration, the equations to be solved are,

$$(A-\lambda_j I)\underline{X} = \underline{V} \quad , \tag{4.3.1}$$

followed by,

$$(A-\lambda_j I)\underline{Y} = \underline{X} \quad , \tag{4.3.2}$$

where A is the N×N tridiagonal centro-symmetric matrix as given in (4.2.1) with N odd or even. Since λ_j is an accurate eigenvalue two iterations should be sufficient to obtain a vector of the required accuracy as indicated by Wilkinson (1962).

The equations above are solved by performing a Gaussian elimination process with partial pivoting (to ensure numerical stability) on the L.H.S. of equation (4.3.1), storing the resultant matrix (A say), and remembering all operations performed on the original matrix (A-λI) i.e. interchanges and elimination factors.

Wilkinson (1965) has shown that $\underline{V}$ can now be written as,

$$\underline{V} = [v_1, v_2, \ldots\ldots, v_N] \quad ,$$

and initially, $$\underline{V} = [1,1,\ldots\ldots\ldots,1] \quad , \tag{4.3.3}$$

and a simple back substitution through the matrix $\bar{A}$ determines the vector X. Now a record of the necessary interchanges can be stored by flags and the required operations can be performed on the vector $\underline{X}$, (i.e. forward substitution), and a further back substitution through $\bar{A}$ gives $\underline{Y}$, the required eigenvector.

It should now be noted that the eigenvectors of a centro-symmetric tridiagonal matrix are either symmetric or anti-symmetric, and take one of the following two forms:

for N even $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N/2} \\ x_{N/2} \\ \vdots \\ x_2 \\ x_1 \end{bmatrix}, \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N/2} \\ -x_{N/2} \\ \vdots \\ -x_2 \\ -x_1 \end{bmatrix}$, and for N odd $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\frac{N-1}{2}} \\ x_{\frac{N-1}{2}+1} \\ x_{\frac{N-1}{2}} \\ \vdots \\ x_2 \\ x_1 \end{bmatrix}, \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\frac{N-1}{2}} \\ x_{\frac{N-1}{2}+1} \\ -x_{\frac{N-1}{2}} \\ \vdots \\ -x_2 \\ -x_1 \end{bmatrix}$ (4.3.4)

As the eigenvectors are symmetric (or antisymmetric) about the centre point the elimination procedure can be performed from both ends of the matrix at once, and as both halves are identical only half of the original matrix need be used. However one element in the other half of the vector must be determined to indicate whether a given vector is symmetric or antisymmetric.

Hence the variables are eliminated in their natural order but the pivotal row is selected at each stage, say the $(i-1)^{th}$ (where

$(i-1) < \frac{(N-1)}{2} - 1$, for N odd, and $(i-1) < \frac{N}{2} - 1$, for N even), as the row having the maximum coefficients of the element x_{i-1}.

At this stage the modified matrix $\overline{(A-\lambda_j I)}$ would have the form,

$$\overline{(A-\lambda_j I)} = \begin{bmatrix} \bar{c}_1 & \bar{b}_2 & \bar{d}_3 & & & & & & \\ 0 & \bar{c}_2 & \bar{b}_3 & \ddots & & & & & \\ & \ddots & \ddots & \ddots & & & 0 & & \\ & & 0 & \bar{c}_{i-1} & \bar{b}_i & \bar{d}_{i+1} & & & \\ & & & b_i & c_i-\lambda & b_{i+1} & \ddots & & \\ & & & & b_{i+1} & c_i-\lambda & b_i & & \\ & & 0 & & d_{i+1} & \bar{b}_i & \bar{c}_{i-1} & 0 & \\ & & & & & \ddots & \ddots & \ddots & 0 \\ & & & & & & \bar{d}_3 & \bar{b}_2 & \bar{c}_1 \end{bmatrix} , \quad (4.3.5)$$

where a bar above an element indicates a possible change (due to the elimination and interchange process).

At each stage of the computation there are only two rows to consider, the $(i-1)^{th}$ reduced row and the i^{th} row as yet unchanged.

The pivotal row is,

$$\bar{c}_{i-1} x_i + \bar{b}_i x_{i+1} + d_{i+1} x_{i+2} , \quad (4.3.6)$$

and the i^{th} row is,

$$b_i x_i + (c_i-\lambda_j) x_{i+1} + b_{i+1} x_{i+2} . \quad (4.3.7)$$

If $|b_i| > |\bar{c}_{i-1}|$ then the two rows are interchanged by exchanging the coefficients of the x_k, k=i,i+1,i+2,

$$\bar{c}_{i-1} \leftrightarrow b_i , \qquad \bar{b}_i \leftrightarrow c_i-\lambda_j , \qquad d_{i+1} \leftrightarrow b_{i+1} ,$$

and noting the interchange. If no interchange occurs this also must be noted. When the variable in the i^{th} row is eliminated by adding $-b_i/\bar{c}_{i+1}$ times the $(i-1)^{th}$ row to the i^{th} row, and this factor is also noted. Since only half the original matrix is being used care must be taken with the centre elements and the even and odd cases

must be considered seperately.

The case when N is even

The matrix (A-λI) is even with centre elements detailed for s=N/2

$$(A-\lambda_j I) = \begin{bmatrix} c_1-\lambda_j & b_2 & & & & & & & \\ b_2 & c_2-\lambda_j & b_3 & & & & & & \\ & b_3 & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & b_s & & & & \\ & & & b_s & c_s-\lambda_j & b_{s+1} & & & \\ & & & & b_{s+1} & c_s-\lambda_j & \ddots & & \\ & & & & & \ddots & \ddots & b_3 & \\ & & & & & & b_3 & c_2-\lambda_j & b_2 \\ & & & & & & & b_2 & c_1-\lambda_j \end{bmatrix} . \qquad (4.3.8)$$

If the Gaussian elimination process is now performed with partial pivoting for the first and last s rows of the matrix, (4.3.8) now becomes,

$$\overline{(A-\lambda_j I)} = \begin{bmatrix} \bar{c}_1 & \bar{b}_2 & d_3 & & & & & & \\ 0 & \bar{c}_2 & \bar{b}_3 & & & & & & \\ & \bar{c}_3 & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & \bar{c}_{s-1} & \bar{b}_s & d_{s+1} & & \\ & & & 0 & \bar{c}_s & \bar{b}_{s+1} & & & \\ & & & & b_{s+1} & \bar{c}_s & 0 & & \\ & & & & d_{s+1} & \bar{b}_s & \bar{c}_{s-1} & \ddots & \\ & & & & & & \ddots & \bar{c}_3 & \\ & & & & & & & \bar{b}_3 & \bar{c}_2 & 0 \\ & & & & & & & d_3 & \bar{b}_2 & \bar{c}_1 \end{bmatrix} . \qquad (4.3.9)$$

This leaves the centre elements to be eliminated as,

$$\begin{pmatrix} \bar{c}_s & \bar{b}_{s+1} \\ \bar{b}_{s+1} & \bar{c}_s \end{pmatrix} . \qquad (4.3.10)$$

The rows may be interchanged, the interchange noted, and the elimination then takes place with the centre elements (4,3,10) becoming,

$$\begin{pmatrix} \bar{\bar{c}}_s & \bar{\bar{b}}_{s+1} \\ 0 & \bar{\bar{c}}_{s+1} \end{pmatrix} \quad . \tag{4.3.11}$$

As the top and bottom halves of the matrix are the same in reverse image form, except for the centre elements after elimination only the upper half of the reduced matrix given below (G say) need be considered

$$G = \begin{bmatrix} \bar{\bar{c}}_1 & \bar{b}_2 & d_3 & & & & \\ & \bar{c}_2 & \bar{b}_3 & & 0 & & \\ & & \bar{c}_3 & & & & \\ & & & \ddots & & & \\ & & & & \bar{c}_{s-1} & \bar{b}_s & d_{s+1} \\ & & 0 & & & \bar{\bar{c}}_s & \bar{\bar{b}}_{s+1} \\ & & & & & & \bar{\bar{c}}_{s+1} \end{bmatrix} \quad . \tag{4.3.12}$$

This half matrix can now be used with corresponding reduced vectors,

$$\left.\begin{aligned} \underline{\bar{X}} &= (x_1, x_2, \ldots\ldots x_s, x_{s+1}) \quad , \\ \underline{\bar{V}} &= (v_1, v_2, \ldots\ldots v_s, v_{s+1}) \quad , \end{aligned}\right\} \tag{4.3.13}$$

to complete the backward substitution. The vector of the full matrix is then,

$$\underline{X} = [x_1, \ldots, x_s, x_{s+1}, zx_{s-1}, \ldots\ldots, zx_1] \quad , \tag{4.3.14}$$

$$\left(z = \left\{ \begin{array}{l} 1 \text{ if } x_x x_{s+1} \text{ is positive or } 0 \\ -1 \text{ otherwise} \end{array} \right\} \right) \quad .$$

The case when N is odd

The matrix $(A-\lambda_j I)$ is given with centre elements detailed for $s=(N-1)/2$.

The case when N is odd

The matrix $(A-\lambda_j I)$ is given with centre elements detailed for $s=(N-1)/2$,

$$(A-\lambda_j I) = \begin{bmatrix} c_1-\lambda_j & b_2 & & & & & & & \\ b_2 & c_2-\lambda_j & b_3 & & & & & & \\ & b_3 & & b_s & & & & & \\ & & & c_s-\lambda_j & b_{s+1} & & & & \\ & & & b_{s+1} & c_{s+1}-\lambda_j & b_{s+1} & & & \\ & & & & b_{s+1} & c_s-\lambda_j & & & \\ & & & & & b_s & & b_3 & \\ & & & & & & & c_2-\lambda_j & b_2 \\ & & & & & & & b_2 & c_1-\lambda_j \end{bmatrix} \qquad (4.3.15)$$

If the Gaussian elimination process is now performed with partial pivoting for the first and last s rows of the matrix, (4.3.15) becomes,

$$(A-\lambda_j I) = \begin{bmatrix} \bar{c}_1 & \bar{b}_2 & d_3 & & & & & & \\ 0 & \bar{c}_2 & \bar{b}_3 & & & & & & \\ & & & & & & & & \\ & & 0 & \bar{c}_{s-1} & \bar{b}_s & d_{s+1} & & & \\ & & & 0 & \bar{c}_s & \bar{b}_{s+1} & & & \\ & & & & b_{s+1} & c_{s+1}-\lambda_j & b_{s+1} & & \\ & & & & & \bar{b}_{s+1} & \bar{c}_s & Q & \\ & & & & & d_{s+1} & & & \\ & & & & & & & \bar{b}_3 & \bar{c}_s & 0 \\ & & & & & & & d_3 & \bar{b}_2 & \bar{c}_1 \end{bmatrix} \qquad (4.3.16)$$

This leaves the centre elements to be eliminated,

$$\begin{pmatrix} \bar{c}_s & \bar{b}_{s+1} & 0 \\ b_{s+1} & c_{s+1}-\lambda_j & b_{s+1} \\ 0 & \bar{b}_{s+1} & \bar{c}_s \end{pmatrix} \tag{4.3.17}$$

The rows may be interchanged, the interchange noted, and the elimination performed for the two remaining rows with the centre elements becoming,

$$\begin{pmatrix} \bar{\bar{c}}_s & \bar{\bar{b}}_{s+1} & d_{s+2} \\ 0 & \bar{\bar{c}}_{s+1} & \bar{\bar{b}}_{s+2} \\ 0 & 0 & \bar{\bar{c}}_{s+2} \end{pmatrix} \tag{4.3.18}$$

As the top and bottom halves of the matrix are the same, in reverse image form except for the centre elements, after the elimination process, only the upper half of the reduced matrix given below (G say) need be considered.

$$G = \begin{bmatrix} \bar{\bar{c}}_1 & \bar{b}_2 & d_3 & & & & & \\ & \bar{c}_2 & \bar{b}_3 & & & & & \\ & & \bar{c}_3 & & & & & \\ & & & \ddots & & & & \\ & & & & \bar{c}_{s-1} & \bar{b}_s & d_{s+1} & \\ & & & & & \bar{\bar{c}}_s & \bar{\bar{b}}_{s+1} & d_{s+2} \\ & & & & & & \bar{\bar{c}}_{s+1} & \bar{\bar{b}}_{s+2} \\ & & & & & & & \bar{\bar{c}}_{s+2} \end{bmatrix} \tag{4.3.19}$$

This half matrix can now be used with corresponding reduced vectors,

$$\left.\begin{aligned} \bar{X} &= (x_1, x_2, \ldots\ldots, x_s, x_{s+1}, x_{s+2}) \quad , \\ \bar{V} &= (v_1, v_2, \ldots\ldots, v_s, v_{s+1}, v_{s+2}) \quad , \end{aligned}\right\} \tag{4.3.20}$$

to complete the back substitution. The vector of the full matrix is then,

$$\underline{X} = (x_1, \ldots\ldots, x_s, x_{s+1}, x_{s+2}, zx_{s-1}, \ldots\ldots, zx_1) \quad , \tag{4.3.21}$$

$$\left(z = \begin{cases} 1 \text{ if } x_s\, x_{s+2} \geq 0 \\ -1 \text{ otherwise} \end{cases} \right)$$

The back substitution through G to obtain $\bar{X}$ with the vector V as given in (4.3.3), is described in algorithmic form below.

When N is even and s=N/2

$$\left.\begin{aligned} x_{s+1} &= v_{s+1}/\bar{\bar{c}}_{s+1} \quad , \\ x_s &= (v_s - \bar{\bar{b}}_{s+1}\bar{\bar{x}}_{s+1})/\bar{\bar{c}}_s \quad , \\ x_i &= (v_i - d_{i+2}x_{i+2} - \bar{b}_{i+1}x_{i+1})/\bar{c}_i \quad , \; i=s-1,s-2,\ldots,1 \end{aligned}\right\} \tag{4.3.22}$$

When N is odd and s=(N-1)/2

$$\left.\begin{aligned} x_{s+2} &= v_{s+2}/\bar{\bar{c}}_{s+2} \quad , \\ x_{s+1} &= (v_{s+1} - \bar{\bar{b}}_{s+2}x_{s+2})/\bar{\bar{c}}_{s+1} \quad , \\ x_s &= (v_s - d_{s+2}x_{s+2} - \bar{\bar{b}}_{s+1}x_{s+1})/\bar{\bar{c}}_s \quad , \\ x_i &= (v_i - d_{i+2}x_{i+2} - \bar{b}_{i+1}x_{i+1})/\bar{c}_i \quad , \; i=s-1,s-2,\ldots,1 \end{aligned}\right\} \tag{4.3.23}$$

Now that the vector $\bar{X}$ has been obtained it can be overwritten on the vector V, and the remembered operations performed on V. The back substitution given in (4.3.22) or (4.3.23) can again be performed and the final vector $\bar{X}$ can now be used to obtain the required eigenvector by using equation (4.3.14) or (4.3.21). The algorithm to carry out the above procedure is given in Appendix 1 in program 7.

4.4 RESULTS

The test matrix used is given in Gregory and Karney (1969). Let $A = [a_{i,j}]$ be the N×N tridiagonal matrix whose elements are defined by

$$\left.\begin{aligned} a_{i,i} &= -[(2i-1)(N-1)-2(i-1)^2] \quad , i = 1,2,\ldots N, \\ a_{i,i+1} &= i(N-i), \; i = 1,2,\ldots N-1, \\ a_{i,i-1} &= (i-1)(N+1-i) \quad , i = 1,3,\ldots N, \\ a_{i,j} &= 0 \quad , \quad \text{if } |i-j| > 1, \text{ for } i,j=1,2,\ldots,N. \end{aligned}\right\} \tag{4.4.1}$$

So that a 6×6 matrix has the form,

$$\begin{bmatrix} -5 & 5 & & & & \\ 5 & -13 & 8 & & 0 & \\ & 8 & -17 & 9 & & \\ & & 9 & -17 & 8 & \\ & 0 & & 8 & -13 & 5 \\ & & & & 5 & -5 \end{bmatrix} \tag{4.4.2}$$

Theoretically the eigenvalues are given by,

$$\lambda_i = -(i-1)i \quad , \qquad i=1,2,\ldots,N \tag{4.4.3}$$

The eigenvectors $x_j^{(i)}$ corresponding to each λ_i are given by,

$$x_j^{(i)} = \frac{1}{\binom{N-1}{i-1}} \sum_{k=0}^{q} (-1)^k \binom{N-k-1}{N-i} \binom{j-1}{k} \binom{j-1+k}{k} \ , \tag{4.4.4}$$

for $j=1,2,\ldots,N$ and $q=\min(i,j)$.

and the symbol $\binom{i}{j}$ denotes the binomial coefficient.

The value of N was chosen to be 10 for the numerical experiments. The results are given in Table (4.4.1) with the eigenvalue at the head of its corresponding eigenvector. The time taken on the Loughborough University of Technology computer (ICL 1904S) was 5 seconds.

The results obtained in Table (4.4.1) were found to agree to 10 significant figures with the results given by (4.4.2) and (4.4.3).

Next the program was tested for a matrix of the same form as (4.4.1) but with N equal to 100. Again the results for both eigenvalues and eigenvectors agreed to at least 10 significant figures with those obtained for the theoretical formula.

Eigenvalues	$-1.4873125*10^{-1}$	-2.000000000	-6.000000000	$-1.200000000*10^{1}$	$-2.000000000*10^{1}$
Corresponding Eigenvectors	$3.162277660*10^{-1}$	$4.954336943*10^{-1}$	$5.222329679*10^{-1}$	$4.534251929*10^{-1}$	$-3.367809164*10^{-1}$
	$3.162277660*10^{-1}$	$3.853373178*10^{-1}$	$1.740776559*10^{-1}$	$-1.511417310*10^{-1}$	$4.113766756*10^{-1}$
	$3.162277660*10^{-1}$	$2.752409413*10^{-1}$	$-8.703882800*10^{-2}$	$-3.778543275*10^{-1}$	$3.178819766*10^{-1}$
	$3.162277660*10^{-1}$	$1.651445648*10^{-1}$	$-2.611164840*10^{-1}$	$-3.346709757*10^{-1}$	$-5.609681940*10^{-2}$
	$3.162277660*10^{-1}$	$5.504818825*10^{-2}$	$-3.481553119*10^{-1}$	$-1.295500551*10^{-1}$	$-3.365809164*10^{-1}$
	$3.162277660*10^{-1}$	$-5.504818825*10^{-2}$	$-3.481553119*10^{-1}$	$1.295500551*10^{-1}$	$-3.365809164*10^{-1}$
	$3.162277660*10^{-1}$	$-1.657445648*10^{-1}$	$-2.611164840*10^{-1}$	$3.346709757*10^{-1}$	$-5.609681940*10^{-2}$
	$3.162277660*10^{-1}$	$-2.752409413*10^{-1}$	$-8.703882800*10^{-2}$	$3.778543275*10^{-1}$	$-3.178819766*10^{-1}$
	$3.162277660*10^{-1}$	$-3.853373178*10^{-1}$	$1.740776559*10^{-1}$	$1.511417310*10^{-1}$	$4.113766756*10^{-1}$
	$3.162277660*10^{-1}$	$-4.954336943*10^{-1}$	$5.222329679*10^{-1}$	$-4.534251929*10^{-1}$	$-3.365809164*10^{-1}$

TABLE 4.4.1

Eigenvalues	$-3.000000000*10^{1}$	$-4.200000000*10^{1}$	$-5.6000000000*10^{1}$	$-7.2000000000*10^{1}$	$-9.0000000000*10^{1}$
Corresponding Eigenvectors	$-2.148344622*10^{-1}$	$-1.167748416*10^{-1}$	$5.269378644*10^{-2}$	$-1.869893980*10^{-2}$	$-4.535159052*10^{-3}$
	$5.012804118*10^{-1}$	$4.281744193*10^{-1}$	$-2.751886625*10^{-1}$	$1.308925786*10^{-1}$	$4.081643147*10^{-2}$
	$-3.580574371*10^{-2}$	$-3.892494721*10^{-1}$	$5.035184038*10^{-1}$	$-3.739787960*10^{-1}$	$-1.632657259*10^{-1}$
	$-3.938631807*10^{-1}$	$-2.335496832*10^{-1}$	$-2.459043367*10^{-1}$	$5.235703144*10^{-1}$	$3.809533604*10^{-1}$
	$-2.148344622*10^{-1}$	$3.113995777*10^{-1}$	$-3.278724490*10^{-1}$	$-2.617851572*10^{-1}$	$-5.714300405*10^{-1}$
	$2.148344622*10^{-1}$	$3.113995777*10^{-1}$	$3.278724490*10^{-1}$	$-2.617851572*10^{-1}$	$5.714300405*10^{-1}$
	$3.938631807*10^{-1}$	$-2.335496832*10^{-1}$	$2.459043367*10^{-1}$	$5.235703144*10^{-1}$	$-3.809533604*10^{-1}$
	$3.580574371*10^{-2}$	$-3.892494721*10^{-1}$	$-5.035184038*10^{-1}$	$-3.739787960*10^{-1}$	$1.632657259*10^{-1}$
	$-5.012804118*10^{-1}$	$4.281744193*10^{-1}$	$2.751786625*10^{-1}$	$1.308925786*10^{-1}$	$-4.081643147*10^{-1}$
	$2.148344622*10^{-1}$	$-1.167748416*10^{-1}$	$-5.269378644*10^{-2}$	$-1.869893980*10^{-2}$	$4.535159052*10^{-3}$

4.5 USE OF THE STURM SEQUENCE ALGORITHM FOR GENERAL TRIDIAGONAL MATRICES USING PARALLEL PROCESSING

If A is the symmetric tridiagonal (N×N) matrix of general form given by,

$$A = \begin{bmatrix} c_1 & b_2 & & & & \\ b_2 & c_2 & b_3 & & 0 & \\ & b_3 & c_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & b_N \\ & 0 & & & b_N & c_N \end{bmatrix} , \qquad (4.5.1)$$

then a similar technique to that described in section 4.3 can be used to carry out an elimination procedure in the matrix $(A-\lambda I)$ from both ends of the diagonal simultaneously using two processors. Care must be taken when dealing with the centre elements, but the sequence $q_i(\lambda)$, $i=1,2,\ldots,N$ so produced can be shown as in Section 4.2 to have the same values as the Sturm sequence of a similar matrix with selected columns and rows interchanged.

By using the Gaussian elimination process the sequence for the case when N is odd and where,

$$s = (N+1)/2 \quad , \qquad (4.5.2)$$

is as follows,

$$\left.\begin{aligned} q_1(\lambda) &= c_1-\lambda \quad , \\ q_i(\lambda) &= c_i-\lambda-b_i^2/q_{i-1}(\lambda), \qquad i=2,3,\ldots,s-1 \\ q_N(\lambda) &= c_N-\lambda \quad , \\ q_i(\lambda) &= c_i-\lambda-b_{i+1}^2/q_{i+1}(\lambda), \qquad i=N-1,N-2,\ldots,s+1, \\ q_s(\lambda) &= c_s-\lambda-b_s^2/q_{s-1}(\lambda)-b_{s+1}/q_{s+1}(\lambda), \end{aligned}\right\} \qquad (4.5.3)$$

and the case when N is even and where

$$s = N/2 \quad , \qquad (4.5.4)$$

gives the sequence,

$$
\left.
\begin{aligned}
q_1(\lambda) &= c_1-\lambda \ , \\
q_i(\lambda) &= c_i-\lambda-b_i^2/q_{i-1}(\lambda), && i=2,3,\ldots,s, \\
q_N(\lambda) &= c_N-\lambda \ , \\
q_i(\lambda) &= c_i-\lambda-b_{i+1}^2/q_{i+1}(\lambda) \ , && i=N-1,N-2,\ldots,s+2 \\
q_{s+1}(\lambda) &= c_{s+1}-\lambda-b_{s+2}^2/q_{s+2}(\lambda)-b_{s+1}^2/q_s(\lambda) \ ,
\end{aligned}
\right\} \quad (4.5.5)
$$

The sequence of $q_i(\lambda)$, i=1,2,...,N can now be used in a bisection process to isolate the eigenvalues of the matrix A as described in Chapter 2.

The order of the calculation of the $q_i(\lambda)$ i=1,N is given in Figure 1.

Order of Calculation of Elements of Sturm Sequence

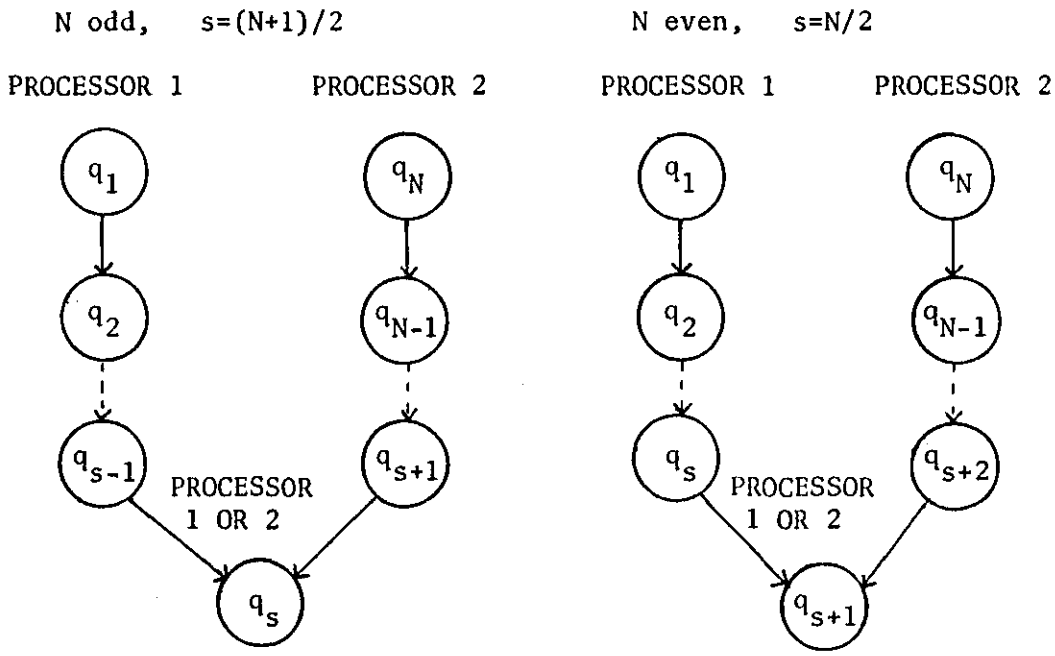


FIGURE 4.5.1

The calculation as in Figure (4.5.1) is performed many times for the different values of λ. The program for calculating the eigenvalues of a symmetric tridiagonal matrix is given in Appendix 1 in program 8.

This program is written in standard ALGOL 60 except that the parallel processing constructs FORK and JOIN have been inserted.

These statements indicate where the program, forks to allow the two processors to work on separate sections independently, and where the program joins as only one processor can then work on the program after collecting results from the sections done in parallel.

Program 8 was run sequentially without fork and join statements on the I.C.L. 1904S at Loughborough University of Technology. The results were identical to those obtained using Program 1 and were arrived at in the same times, which indicates that there was little inefficiency in splitting the sequences up into two halves. Of course there were certain overheads introduced by splitting the sequence into two, but these were negligible compared to the time taken actually computing the eigenvalues.

Next the program was translated into FORTRAN for use on the Loughborough University Interdata parallel computer. This configuration has two model 70 processors sharing a 32K block of core, each also having 32K of private core.

Certain amendments were made to the program to improve its efficiency for running in parallel. For example when the two processors are working and both wish to access different parts of the same array one processor is held up and has to wait until the other has finished. This is known as store clashing, and as much data as possible was put into the private memories to avoid this.

When an efficient program had been obtained (program 9 in the Appendix) it was tested on a 64×64 matrix. (The program was being run for timing comparisons so the actual results are not given here, they were in fact verified correct but the only result required was time taken).

First the program was run sequentially without forks and joins and it took 24.97 seconds to obtain the results. Next it was run

in parallel mode with forks and joins included, and the results were obtained in 21.07 seconds. Therefore the results obtained when running in parallel mode were obtained in 84% of the time taken when the program was run in sequential mode. This result is dissappointing as the parallel method uses more resources for a longer period. During the calculation neither processor is available for other work yet the two processors are only working for 54% and 87% of the total time taken.

An alternative method of using the original sequence produced in section4.4.2 in a parallel fashion was suggested and developed by Barlow (1977a, 1977b). This method is described briefly here.

a) The maximum and minimum values of the eigenvalues are obtained using Gerschgorins theorem.

b) A queue is set up using part a) results to initiate it. Each element in the queue describes the upper and lower bounds of the interval, how many eigenvalues are below the interval and how many eigenvalues above it.

c) This part of the algorithm is done in parallel. As it becomes free a processor takes an interval off the queue, bisects it, and calculates the Sturm sequence for this value, to determine how many eigenvalues there are in each half of the interval. There are now several possibilities to follow:-

1) Eigenvalues are in each half interval, in which case both intervals are put back on the queue, and the processor then chooses the next interval on the queue.

2) Eigenvalues are in only one half of the interval, in this case only the full interval is returned to the queue before the processor moves to the next interval in the queue.

3) The half intervals are less than the desired accuracy. All the eigenvalues in this interval are considered found and assigned

the mid point value. The interval is then removed from the queue and the processor returns to the head of the queue.

4) If an interval is found to contain only one eigenvalue, bisection is continued without recourse to the queue until the eigenvalue is found correct to the required accuracy. The interval is then removed from the queue and the processor returns to the queue to pick up another interval.

The program to perform the above algorithm is given in Barlow (1977a) and some results using this program are given below.

This program was run using the same (64×64) matrix as before from which the accuracy of results could be verified. When run sequentially the program took 20.29 seconds, this is a slight reduction in time of 19% over the original algorithm run sequentially, and a small reduction in time when run in parallel. When the new algorithm was run in parallel the time taken was 11.30 seconds. This indicates that run in parallel the new algorithm takes only 56% of the time taken to run sequentially. This was the order of speed up that was aimed for in the original algorithm.

This indicates that for this type of problem it is not good strategy to obtain information (the Sturm sequence) in a parallel fashion and use an existing algorithm (bisection), but better to obtain the information (the Sturm sequence) and then use it in a parallel manner (Barlow's algorithm) in a new or modified existing algorithm.

Barlow's parallel algorithm has two other major advantages. One is that as the unmodified Sturm sequence is used the parallel algorithm can be used in conjunction with the Sturm sequences of any matrix. Therefore this algorithm can be used to find the eigenvalues of any matrix for which a Sturm sequence exists. The ensuing chapters of

this work contain Sturm sequences for many types of matrix and can be used in conjunction with the algorithm.

The other major advantage of the parallel algorithm is that it can be used in a system with any number of processors. The algorithm is flexible in that it does not require a set number of processors. Also during the computation processors can be put to work on the queue as they become free and taken away as required for other work. The number of processors working on the problem therefore need not remain constant. The maximum number of processors that can be used on the problem without inefficiencies due to waiting for work to become available is equal to the number of elements in the queue. The length of this queue starts at one and gradually increases to a maximum of N and then decreases to zero.

The necessary number of processors can then be switched in as required by looking at the queue length.

This algorithm is thus very flexible and can be of particular use in fields such as meteorology weather forecasting where solutions are required quickly in real time.

4.5 THE CALCULATION OF THE EIGENVECTORS OF A SYMMETRIC TRIDIAGONAL MATRIX BY INVERSE ITERATION

A similar method to that of section 4.3 can now be used on a symmetric tridiagonal matrix. If two processors are available the matrix can be eliminated from both corners at once on the matrix folding principle, with both halves of the resultant matrix being noted as the original matrix is no longer centro-symmetric.

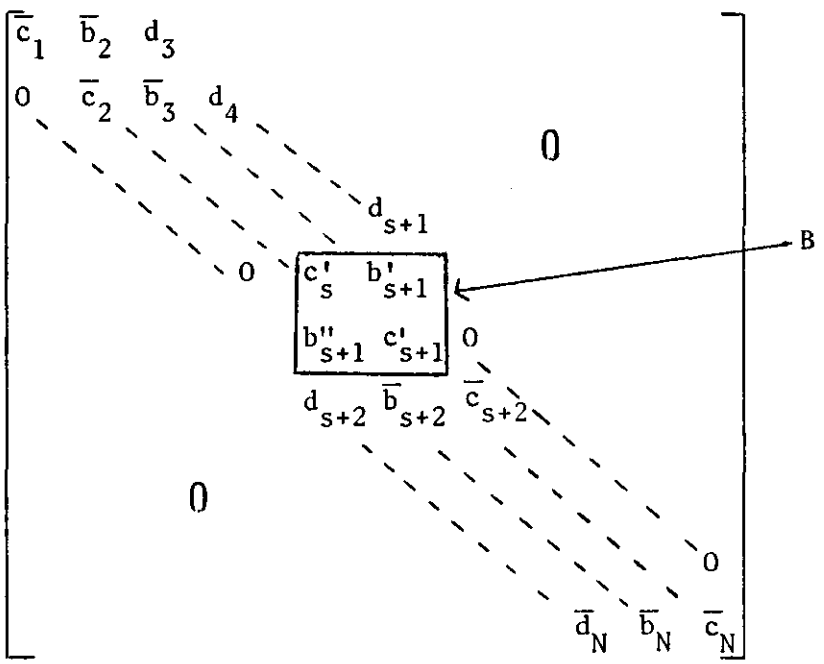
If an eigenvalue λ_j (say) of the matrix A, as given in (4.4.1) has been found, a Gaussian elimination procedure applied to the matrix $(A-\lambda_j I)$ must be considered for N odd and even separately.

The case when N is even

Let

$$s = N/2 \quad , \tag{4.6.1}$$

The Gaussian elimination process with a partial pivoting strategy is now performed on the first and last s rows of the matrix $(A-\lambda_j I)$ simultaneously by each processor. The resultant matrix is,



(4.6.2)

The elimination in the submatrix B is now performed by one processor and the submatrix becomes

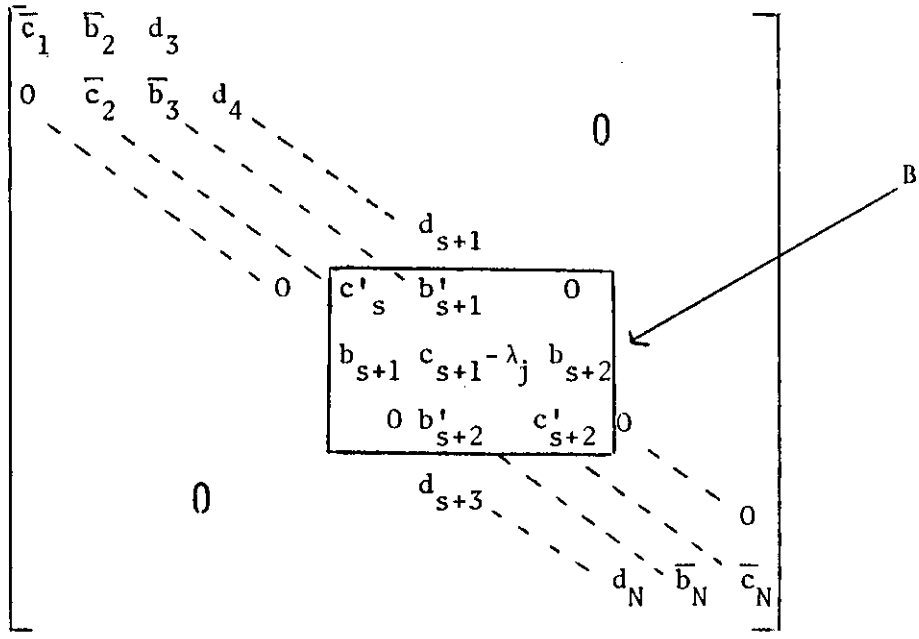
$$\begin{bmatrix} \bar{c}_s & \bar{b}_{s+1} \\ 0 & \bar{c}_{s+1} \end{bmatrix} \quad . \tag{4.6.3}$$

The case when N is odd

Let

$$s = (N-1)/2 \quad . \tag{4.6.4}$$

The Gaussian elimination process with partial pivoting strategy is now performed on the first and last s rows of the matrix $(A-\lambda_j I)$ simultaneously by each processor. The resultant matrix is,



(4.6.5)

The elimination in the submatrix B is now performed by one processor in two stages, the first to eliminate b_{s+1},

$$\begin{pmatrix} \bar{c}_s & \bar{b}_{s+1} & d_{s+2} \\ 0 & c'_{s+1} & b''_{s+2} \\ 0 & b'_{s+2} & c'_{s+2} \end{pmatrix} , \qquad (4.6.6)$$

and the second stage is to eliminate b'_{s+2},

$$\begin{pmatrix} \bar{c}_s & \bar{b}_{s+1} & d_{s+2} \\ & \bar{c}_{s+1} & \bar{b}_{s+2} \\ & & \bar{c}_{s+2} \end{pmatrix} . \qquad (4.6.7)$$

The elimination process is essentially the same as that described in section 4.3. The backward substitution now takes place through both halves of the matrix simultaneously to produce a full vector. The whole process i.e. inverse iteration is then repeated using the new vector to obtain the required eigenvector as given earlier.

If the vector

$$\underline{X} = (x_1, x_2, \ldots, x_N) \qquad , \qquad (4.6.8)$$

is the vector obtained from the back substitution the order of calculation is as in Figure (4.6.1).

Order in which elimination and back substition are performed

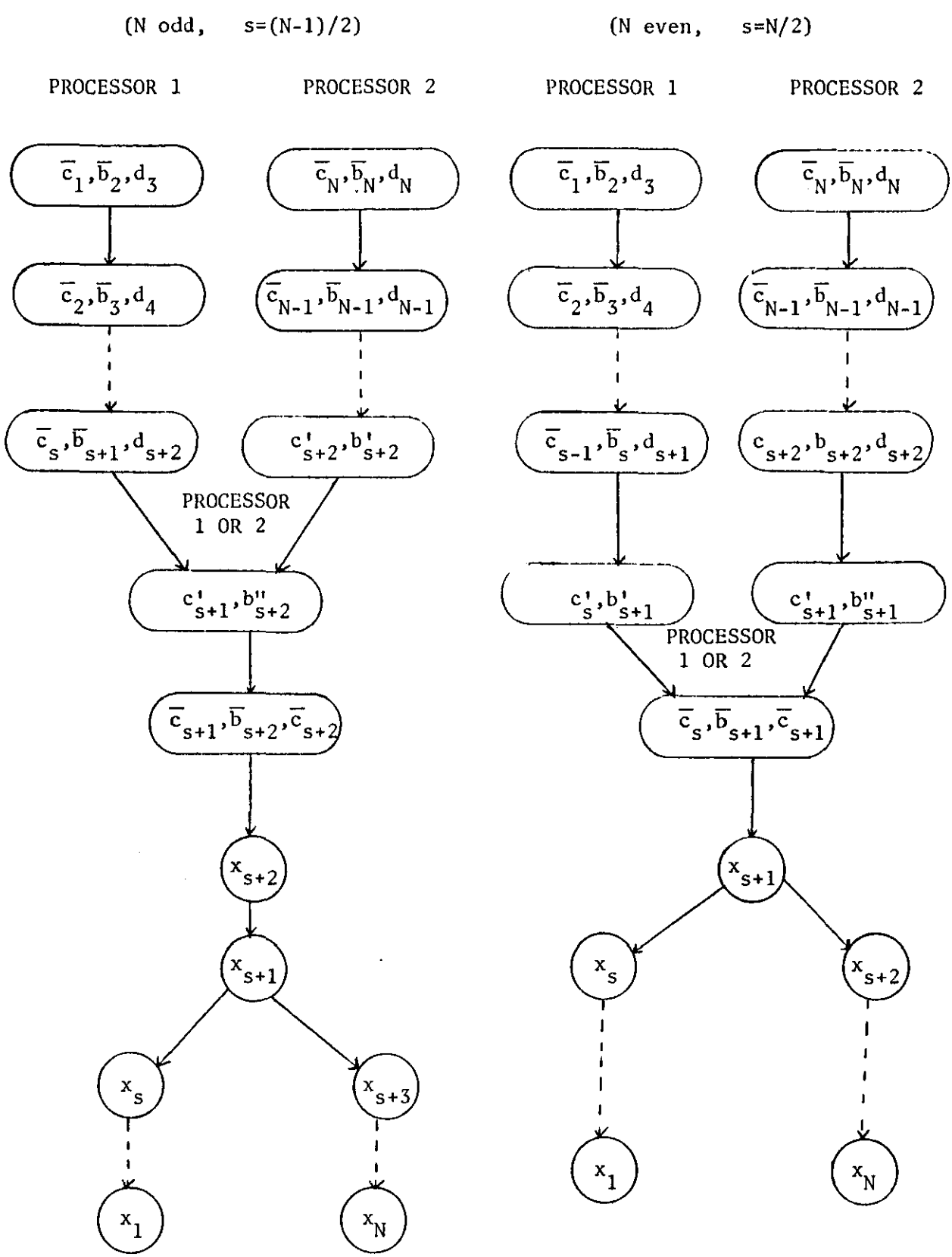


FIGURE 4.6.1

The program to carry out this algorithm is program 10 in Appendix 1. It is given in standard ALGOL 60 with FORK and JOIN statements added to indicate which sections are run in parallel, This program as before was translated into FORTRAN for running on the Interdata system at Loughborough University of Technology. The FORTRAN program is program 11 in the Appendix.

Program 11 was then run using a (64×64) matrix for which the eigenvalues were known, to determine all the eigenvectors on the Interdata parallel computer. When run sequentially without FORK and JOIN statements this took 11.51 seconds. The program was run in parallel fashion using both processors and took 9.25 seconds. The parallel version therefore takes 80% of the time taken to run sequentially. Thus when run in parallel the algorithm uses 60% more resources than when run sequentially to obtain an improvement in time taken of 20%. This inefficiency is due to several factors, 1) store clashing or queuing up to use the same array, 2) overheads in performing FORK and JOIN statements which have been inserted as low level subroutines, 3) one processor always performs the sequential sections and during this time the other stands idle. For these reasons the method is very inefficient and is not recommended.

A simple procedure was adopted to produce a fast efficient algorithm similar in idea to the method described in section 4.5. Each processor was given a copy of the input matrix, and the program to obtain eigenvectors by the normal method. Then each processor was given an eigenvalue and left to find the eigenvector. When a processor has finished it is given another eigenvalue until all vectors are found. On the same test matrix this method was run sequentially using one processor, and took 11.28 seconds. Then it was run in parallel on two processors taking 5.76 seconds. The time taken running in parallel is reduced to 51.1% of the time taken to run sequentially.

This algorithm is preferred as it is very efficient, having small losses in overheads for parallel running. Also it is very flexible because any form of inverse iteration can be used in the method, and any number of processors can be utilised working out as many eigenvectors in parallel as there are processors available.

The results of sections 4.5 and 4.6 indicate the best approach to solving problems of the type covered in this chapter. In general, it is not a good strategy to modify a well known efficient method of this type (viz. obtaining Sturm sequence, performing inverse iteration) for use in a parallel fashion, as this will cause too many inefficiencies and add too many constraints to the method to make it worthwhile. The inefficiencies are caused by parallel implementation, and waiting while single processor sections are performed, and the constraints are that only algorithms for which a parallel implementation can be found can be used, and these methods will have inherently a fixed number of processors necessary to perform the calculation. Rather it is a better strategy to use a proven method and try and restructure this in some parallel fashion, as has been shown in sections 4.5, and 4.6. These results are in agreement with some of the comments made by Stone (Traub, 1973) and possibly indicate the most fruitful areas for future research in restructuring algorithms suitable for parallel processing.

More detailed figures and analysis of the results obtained in sections 4.5 and 4.6 are given by Barlow (1977a, 1977b).

CHAPTER 5

THE NUMERICAL CALCULATION OF THE EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC SPARSE QUINDIAGONAL MATRIX

5.1 INTRODUCTION

In this chapter is described the formulation of an algorithm to determine implicitly the Sturm sequence for a symmetric sparse quindiagonal matrix of semi-bandwidth p. The sequence is then used in a bisection process to isolate and determine the eigenvalues of the matrix. Because of the unique form of the sequence a modification to the bisection algorithm is suggested which shows marked improvements in time taken when used together on a computer.

An inverse iteration process initially modified to cope with the sparsity of the matrix is then derived to determine the eigenvectors of the matrix. Each of the algorithms described are formulated in a manner to economise on the use of the storage space in the computer memory.

The final section gives the formulation of a Sturm sequence for the unsymmetric sparse quindiagonal matrix of semi-bandwidth p. The sequence is then used in conjunction with Muller's method to obtain the eigenvalues of the matrix. An inverse iteration procedure similar to the method of the previous section can then be used to determine the eigenvectors.

5.2 FORMULATION OF THE STURM SEQUENCE AND THE CALCULATION OF THE EIGENVALUES

The eigenvalues of a N×N matrix are given by a determinental relation which can be expressed as

$$\det\,(C-\lambda I) = 0\ ,$$

which, in full matrix notation, can be written in the form

$$\det \begin{bmatrix} c_1-\lambda & b_2 & & & & & d_p & & 0 \\ b_2 & c_2-\lambda & b_3 & & & 0 & & d_{p+1} & \\ & \ddots & \ddots & \ddots & & & & & d_N \\ & & \ddots & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & b_{N-1} & & & \\ d_p & & & & b_{N-1} & c_{N-1}-\lambda & b_N & & \\ & d_{p+1} & & 0 & & b_N & c_N-\lambda & & \\ 0 & & d_N & & & & & & \end{bmatrix} = 0 \qquad (5.2.1)$$

(p)

Matrices of the form given by C i.e. symmetric, sparse quindiagonal and of semi-bandwidth p occur frequently in vibration and other problems associated with second order partial differential equations.

Consider the two types of electromagnetic wave propogation in a long conducting cylinder of rectangular cross-section. In the transverse magnetic wave the magnetic-field vector has no longitudinal component, while the longitudinal component of the electric-field vector vanishes on the walls of the guide and satisfies the two dimensional equation,

$$-\left(\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}\right) = \lambda\phi \ , \qquad (5.2.1a)$$

throughout the cross-section where λ is a frequency parameter. In the transverse electric wave the longitudinal component of the electric-field vector vanishes, while the longitudinal component of the magnetic-field vector satisfies (5.2.1a). At the walls the normal derivative of the longitudinal magnetic field must be zero.

The problem of determining these modes of propagation requires determination of the functions $\phi(x,y)$ and corresponding eigenvalues λ which satisfy (5.2.1a) with the appropriate boundary conditions. This is given in figure (5.2.1) with

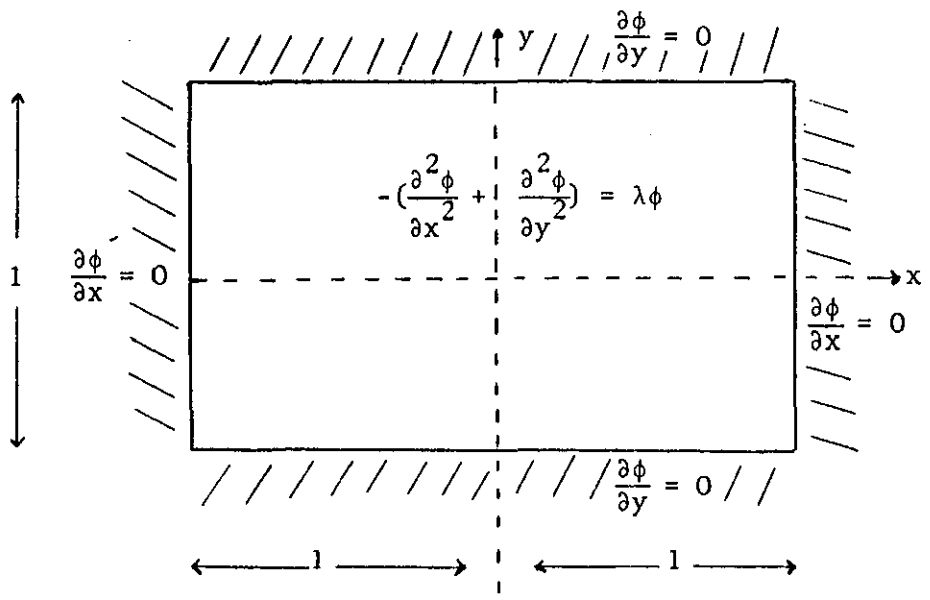


FIGURE 5.2.1

boundary conditions that apply for a transverse electric wave. For a transverse magnetic wave, it is still equation (5.2.1a) that must be solved, but with the boundary conditions,

$$\phi = 0 , \quad \text{on} \quad \begin{cases} x = \pm 1 \\ y = \pm \frac{1}{2} \end{cases} \tag{5.2.1b}$$

Now a grid or lattice can be placed over the waveguide of figure (5.2.1) with a uniform mesh size of h as in Figure 5.2.1, such that $\bar{N}h=1$ and $Mh=2$.

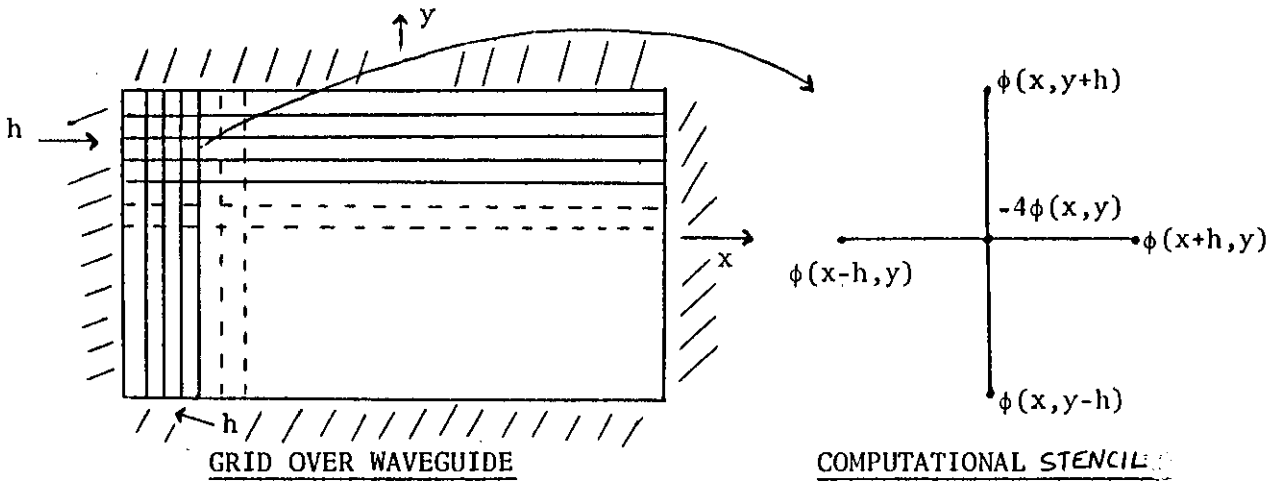


FIGURE 5.2.2

Then if $\phi(x,y)$ is the value of the function at a point on the grid, $\phi(x+h,y)$ is the value of the function at the mesh point to the right (in figure 5.2.2), and $\phi(x,y+h)$ is the value of the function at the mesh point above. Assuming that $\phi(x,y)$ is sufficiently differentiable then by Taylor's theorem

$$\left.\begin{aligned} \phi(x+h,y) &= \phi(x,y) + h\frac{\partial\phi}{\partial x}(x,y) + \frac{h^2}{2}\frac{\partial^2\phi}{\partial x^2}(x,y)+\ldots. \\ \phi(x-h,y) &= \phi(x,y) - h\frac{\partial\phi}{\partial x}(x,y) + \frac{h^2}{2}\frac{\partial^2\phi}{\partial x^2}(x,y)+\ldots. \end{aligned}\right\} \tag{5.2.1c}$$

combining these two equations gives,

$$\frac{\partial^2\phi}{\partial^2 x^2}(x,y) = \frac{\phi(x+h,y)-2\phi(x,y)+\phi(x-h,y)}{h^2} + O(h^4) , \tag{5.2.1d}$$

Similarly for the y direction it can be shown that

$$\frac{\partial^2\phi}{\partial y^2}(x,y) = \frac{\phi(x,y+h)-2\phi(x,y)+\phi(x,y+h)}{h^2} + O(h^4) \qquad (5.2.1e)$$

The truncation error consisting of the remaining small terms of the Taylor expansion. Substituting (5.2.1e) and (5.2.1d) into (5.2.1a) gives the finite difference approximation to the equation at the point (x,y),

$$-(\phi(x+h,y)+\phi(x-h,y)+\phi(x,y+h)+\phi(x,y-h)-4\phi(x,y)) = \lambda\phi(x,y) \qquad (5.2.1f)$$

Equation (5.2.1f) can be applied at all the grid points of figure (5.2.2), taking note of values at the boundary, to give $(\bar{N}\times M)$ simultaneous equations in $\phi(x,y)$ to be solved. By considering the $(\bar{N}\times M)$ equation in column order on the grid they can be expressed in matrix notation as,

$$A\underline{\Phi} = \lambda\underline{\Phi} \quad , \qquad (5.2.g)$$

when,

$$\underline{\Phi}^T = (\phi_{1,1},\phi_{2,1},\ldots\ldots\phi_{N,1},\phi_{1,2},\phi_{2,2},\ldots\ldots,\phi_{N,2},\ldots\ldots\phi_{N,M}), \qquad (5.2.1h)$$

where

$$\phi_{i,j} = \phi(x+ih,y+jh) \ .$$

The matrix A is of order $(\bar{N}M\times\bar{N}M)$ and has the form,

$$A = \begin{bmatrix} D & -I & & & \\ -I & D & -I & & \\ & -I & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & D \end{bmatrix} \quad , \qquad (5.2.1k)$$

where D is the (N×N) matrix,

$$D = \begin{bmatrix} 4 & -1 & & & & & \\ -1 & 4 & -1 & & & 0 & \\ & -1 & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & & -1 \\ & 0 & & & \ddots & -1 & 4 \end{bmatrix} , \qquad (5.2.1\ell)$$

and I is the $\bar{N}\times\bar{N}$ identity matrix. The semi-bandwidth of matrix A is therefore $(\bar{N}+1)$, and it has the same form as matrix C in (5.2.1).

If a Gaussian elimination process is performed on the matrix $(C-\lambda I)$ without using a pivoting strategy then, after the first stage of reduction, the matrix becomes,

$$\begin{bmatrix} c_1-\lambda & b_2 & & & d_p & & \\ 0 & c_2-\lambda-b_2^2/(c_1-\lambda) & b_3 & & -b_2d_p/(c_1-\lambda) & d_{p+1} & 0 \\ & b_3 & c_3-\lambda & & 0 & & d_N \\ & & & \ddots & & \ddots & \\ 0 & -d_pb_2/(c_1-\lambda) & & & c_p-\lambda-d_p^2/(c_1-\lambda) & & \\ & d_{p+1} & 0 & & & \ddots & c_{N-1}-\lambda \; b_N \\ 0 & & & \ddots & & & b_N \; , \; c_N-\lambda \\ & & & d_N & & & \end{bmatrix}$$

(5.2.2)

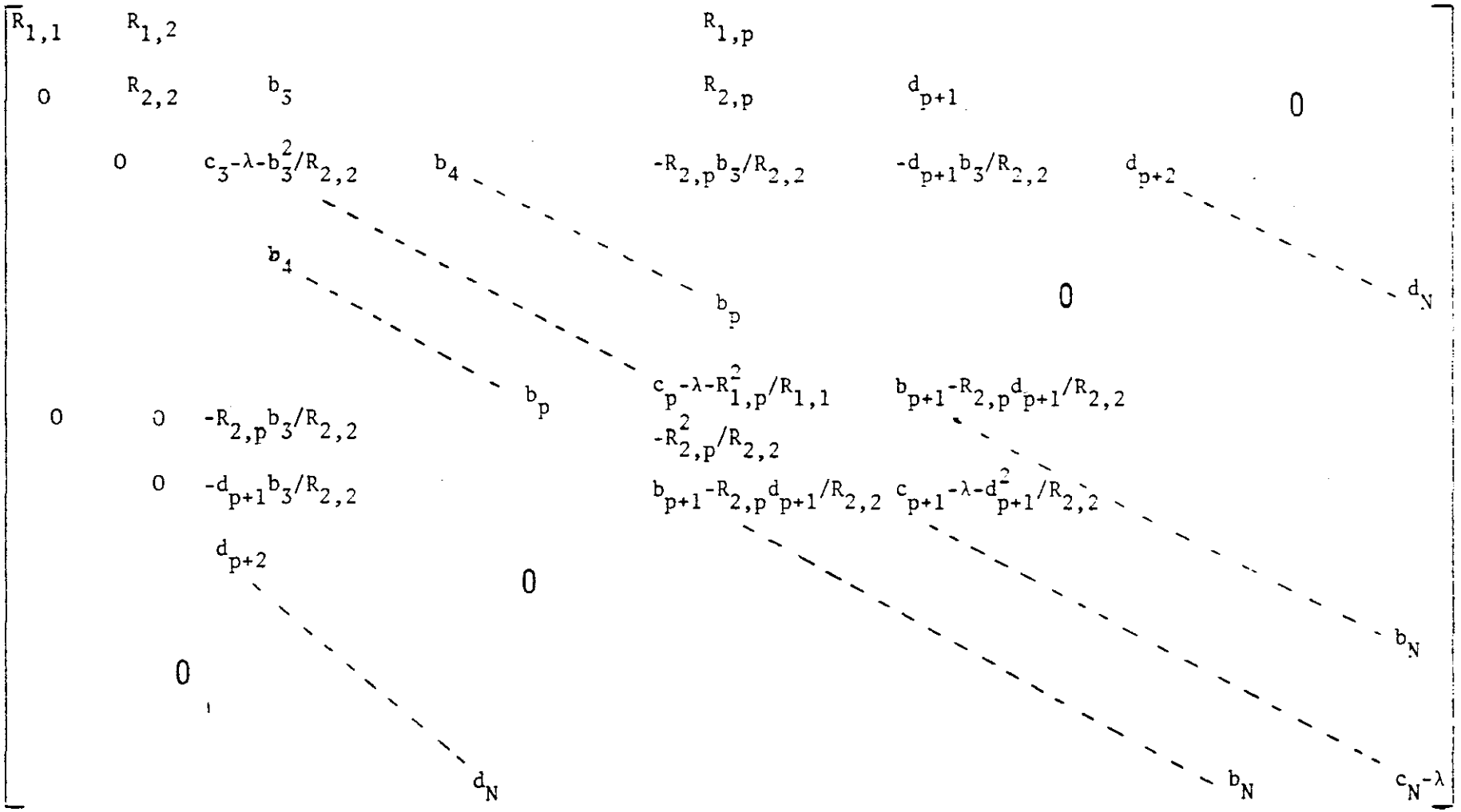
Certain elements of the reduced matrix can now be relabelled in the following fashion:

$$\left.\begin{aligned} R_{1,1} &= c_1-\lambda \; , \qquad R_{1,2} = b_2 \; , \qquad R_{1,p} = d_p \; , \\ R_{2,2} &= c_2-\lambda-R_{1,2}^2/R_{1,1} \; , \\ R_{2,p} &= -b_2d_p/(c_1-\lambda) = -R_{1,2}R_{1,p}/R_{1,1} \; . \end{aligned}\right\} \qquad (5.2.3)$$

If the relabelled elements are included in the reduced matrix, (5.2.2) can now be written as

$$\det \begin{bmatrix} R_{1,1}, & R_{1,2}, & & & R_{1,p} & & 0 \\ 0 \ , & R_{2,2}, & b_3 & & R_{2,p}, & d_{p+1} & \\ & b_3, & c_3-\lambda & & & & \\ & & & 0 & & & d_N \\ & & & & c_p-\lambda-R_{1,p}^2/R_{1,1} & & \\ & & & & & & \\ 0 \ , & R_{2,p} & & 0 & & c_{N-1}-\lambda, & b_N \\ & d_{p+1} & & & & & \\ & 0 & & d_N & & b_N & c_N-\lambda \end{bmatrix} = 0 \ , \qquad (5.2.4)$$

The next stage of Gaussian elimination is performed and (5.2.4) becomes,

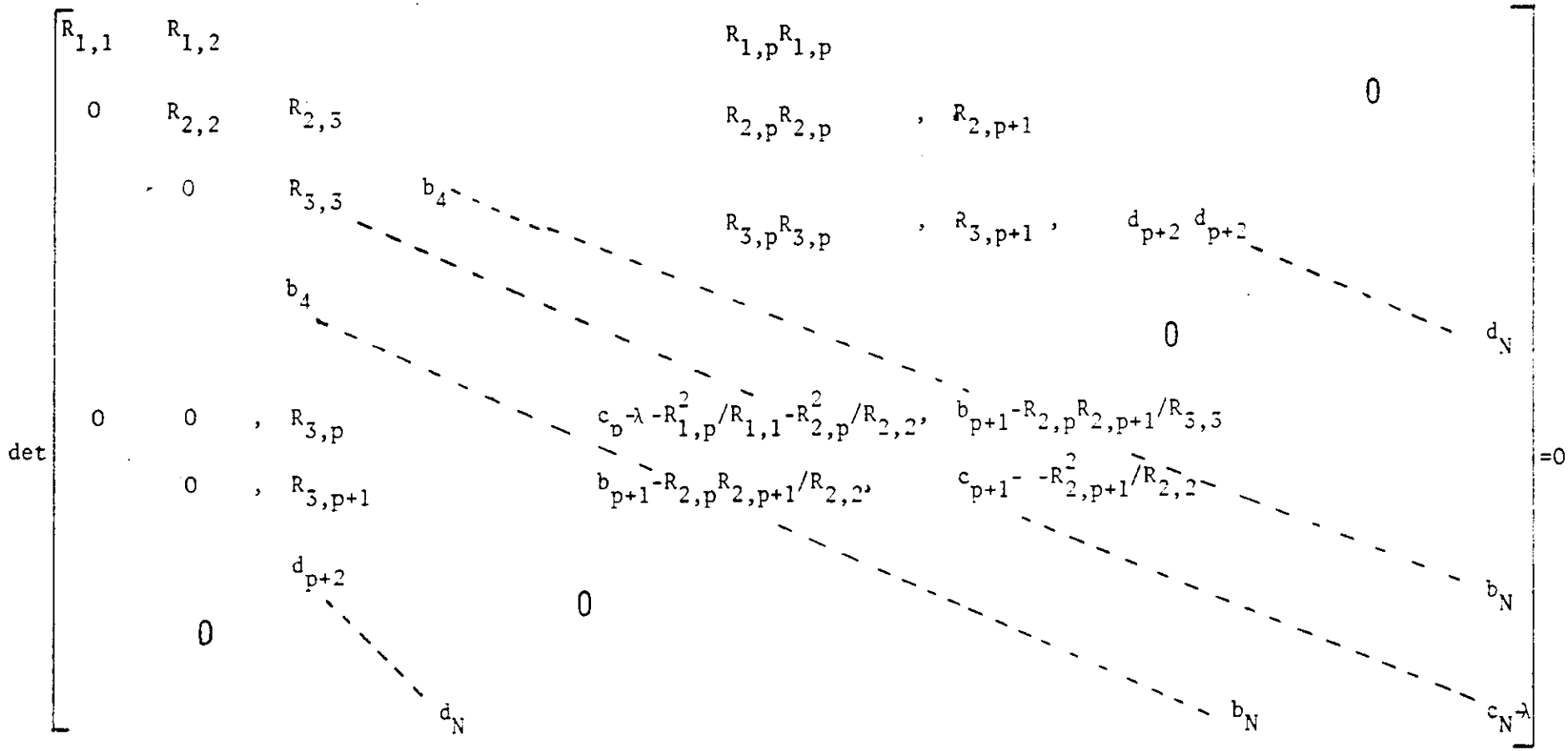


(5.2.5)

Again the elements of the matrix that will not be affected by further steps of the reduction process, except by elimination, can be relabelled as follows:-

$$\left.\begin{aligned} &R_{2,3} = b_3 \ , \qquad R_{2,p+1} = d_{p+1} \ , \\ &R_{3,p} = -R_{2,p}b_3/R_{2,2} = -R_{2,p}R_{2,3}/R_{2,2}, \ R_{3,p+1}=-d_{p+1}b_3/R_{2,2}=R_{2,p+1}R_{2,3}/R_{2,2} \\ &R_{3,3} = c_3-\lambda-b_3^2/R_{2,2} = c_3-\lambda-R_{2,3}^2/R_{2,2} \end{aligned}\right\} \qquad (5.2.6)$$

If these relabelled elements are included in the reduced matrix, (5.2.5) can now be written as:-



$$
\det\begin{bmatrix}
R_{1,1} & R_{1,2} & & & & R_{1,p}R_{1,p} & & & 0 \\
0 & R_{2,2} & R_{2,3} & & & R_{2,p}R_{2,p} & , & R_{2,p+1} & \\
 & 0 & R_{3,3} & b_4 & & R_{3,p}R_{3,p} & , & R_{3,p+1} , & d_{p+2}\ d_{p+2} \\
 & & b_4 & & & & 0 & & \\
0 & 0 & , & R_{3,p} & c_p-\lambda-R_{1,p}^2/R_{1,1}-R_{2,p}^2/R_{2,2}, & & b_{p+1}-R_{2,p}R_{2,p+1}/R_{3,3} & & \\
 & 0 & , & R_{3,p+1} & b_{p+1}-R_{2,p}R_{2,p+1}/R_{2,2}, & & c_{p+1}-\ -R_{2,p+1}^2/R_{2,2} & & \\
 & & d_{p+2} & & 0 & & & & \\
 & 0 & & & & & & & \\
 & & & d_N & & & b_N & & c_N-\lambda
\end{bmatrix} = 0
$$

(5.2.7)

The next stage of Gaussian elimination can now be performed and (5.2.7) becomes:-

$$
\begin{bmatrix}
R_{1,1} & R_{1,2} & & & & R_{1,p} & & & \\
0 & R_{2,2} & & & & R_{2,p} & R_{2,p+1} & & 0 \\
 & 0 & c_3-\lambda-b_3^2/R_{2,2} & b_4 & & R_{3,p} & R_{3,p+1} & d_{p+2} & \\
 & & 0 & c_4-\lambda-b_4^2/R_{3,3} & \ddots & -R_{3,p}b_4/R_{3,3} & -R_{3,p+1}b_4/R_{3,3} & -b_4d_{p+2}/R_{3,3} & \ddots \\
 & & & b_5 & \ddots & & & & d_N \\
 & & & & & b_p & & & \\
 & & & & & & 0 & & \\
 & & & & & c_p-\lambda-R_{1,p}^2/R_{1,1} & b_{p+1}-R_{2,p}R_{2,p+1}/R_{2,2} & -d_{p+2}R_{3,p}/R_{3,3} & \\
0 & 0 & 0 & -R_{3,p}b_4/R_{3,3} & & R_{2,p}^2/R_{2,2}-R_{3,p}^2/R_{3,3} & -R_{3,p+1}R_{3,p}/R_{3,3} & & \\
 & & & & b_p & b_{p+1}-R_{2,p}R_{2,p+1}/R_{2,2} & c_{p+1}-\lambda-R_{2,p+1}^2/R_{2,2} & b_{p+2}-d_{p+2}R_{3,p}/R_{3,3} & \\
0 & 0 & & -R_{3,p+1}b_4/R_{3,3} & & -R_{3,p+1}R_{3,p}/R_{3,3} & -R_{3,p}^2/R_{3,3} & & \\
 & & & & & -d_{p+2}R_{3,p}/R_{3,3} & b_{p+2}-d_{p+2}R_{3,p}/R_{3,3} & c_{p+2}-\lambda-d_{p+2}^2/R_{3,3} & \ddots \\
 & 0 & & -b_4d_{p+2}/R_{3,3} & 0 & & & & b_N \\
 & & & & & & & \ddots & \\
 & & 0 & & & & & b_N & c_N-\lambda \\
 & & & & d_N & & & &
\end{bmatrix}
\tag{5.2.8}
$$

As before the elements of the matrix can be relabelled in the following fashion:-

$$\left.\begin{aligned} &R_{3,4} = b_4 \ , \qquad R_{3,p+2} = d_{p+2} \ , \\ &R_{4,p} = R_{3,p}b_4/R_{3,3} = R_{3,p}R_{3,4}/R_{3,3}, R_{4,p+1}=R_{3,p+1}b_4/R_{3,3}=R_{3,p+1}R_{3,4}/R_{3,3} \\ &R_{4,p+2} = -b_4d_{p+2}/R_{3,3}=-R_{3,4}R_{3,p+2}/R_{3,3}, \ R_{4,4}=c_4-\lambda-b_4^2/R_{3,3}=c_4-\lambda-R_{3,4}^2/R_{3,3} \end{aligned}\right\} \tag{5.2.9}$$

The Gaussian elimination and relabelling is continued as above, and when the elimination is completed after N-1 steps the matrix is of the form,

$$\begin{bmatrix} R_{1,1} & R_{1,2} & & R_{1,p} & & 0 \\ & R_{2,2} & R_{2,3} & R_{2,p} & R_{2,p+1} & \ddots \\ & & R_{3,3} & R_{3,p} & R_{3,p+1} & \ddots \\ & & & \ddots & \ddots & \ddots \\ & & & R_{p-1,p} & \ddots & R_{N-p+1,N} \\ & & & R_{p,p} & \ddots & \vdots \\ & & 0 & & \ddots & R_{N-1,N} \\ & & & & & R_{N,N} \end{bmatrix} \tag{5.2.10}$$

It can easily be seen from (5.2.3), (5.2.6), (5.2.8) and (5.2.9) that at each stage of the elimination the new elements $R_{i,j}$ are defined recursively in terms of the previous elements and elements of the original matrix $(C-\lambda I)$.

From equations (5.2.3), (5.2.6) and (5.2.9) obviously,

$$\left.\begin{aligned} &R_{i,i+1} = b_i \ , \qquad i=2,p-1, \\ &R_{i+1,i+p} = d_{p+i} \ , \qquad i=0,N-p, \end{aligned}\right\} \tag{5.2.11}$$

Also,

$$R_{i,i} = c_i - \lambda - R^2_{i,i-1}/R_{i-1,i-1}, \quad i=2,p-1 \quad . \tag{5.2.12}$$

In equation (5.2.4) the p^{th} element of the main diagonal is,

$$c_p - \lambda - R^2_{1,p}/R_{1,1} \quad . \tag{5.2.13}$$

In equation (5.2.7) the p^{th} element of the main diagonal is now,

$$c_p - \lambda - R^2_{1,p}/R_{1,1} - R^2_{2,p}/R_{2,2} \quad , \tag{5.2.14}$$

and the $p+1^{th}$ element of the main diagonal is,

$$c_{p+1} - \lambda - R^2_{2,p+1}/R_{2,2} \quad . \tag{5.2.15}$$

It can easily be seen from equations (5.2.13), (5.2.14) and (5.2.15) that for any main diagonal elements, including the first p-1 elements, the following relationship is true,

$$R_{i,i} = c_i - \lambda - \sum_{j=1}^{p-i} R^2_{i-p+j,i}/R_{i-p+j,i-p+j}, \quad i=p,p+1,\ldots,N, \tag{5.2.16}$$

Equations (5.2.4) shows that the element b_{p+1} is unaltered after the first step of elimination. After the next step of elimination from equation (5.2.7) it can be seen that b_{p+1} has subtracted from it the term,

$$R_{2,p}R_{2,p+1}/R_{2,2} \quad . \tag{5.2.17}$$

After the next step of elimination b_{p+1} has a further term subtracted, and from equation (5.2.8) becomes,

$$b_{p+1} - R_{2,p}R_{2,p+1}/R_{2,2} - R_{3,p}R_{3,p+1}/R_{3,3} \tag{5.2.18}$$

If the elimination process is continued the final element in the b_{p+1} position is labelled $R_{p,p+1}$ as in (5.2.10).

Following the pattern established in equations (5.2.17) and (5.2.18) the final value for $R_{p,p+1}$ can be written as

$$R_{p,p+1} = b_{p+1} - \sum_{j=1}^{p-2} R_{1+j,p}R_{1+j,p+1}/R_{1+j,1+j} \quad , \tag{5.2.19}$$

If the relationship for several more $R_{i,j}$ elements are established in a similar fashion, the general term for all $R_{i,j}$ elements ($i \leq j$, $j > p$) can easily be recognised. This is given below in the relationships for all $R_{i,j}$ elements occurring in (5.2.10).

$$\left.\begin{aligned}
&R_{1,1} = c_1 - \lambda \ , \\
&R_{i,i-1} = b_i \ , \quad R_{i,i} = c_i - \lambda - R^2_{i-1,i}/R_{i-1,i-1}, \quad i=2,3,\ldots p-1, \\
&R_{i+1,i+p} = d_{p+i}, \qquad\qquad i=0,1,\ldots N-p \\
&R_{k,i} = \ell_{k,i} - \sum_{j=1}^{p+k-i-1} R_{i-p+j,k} R_{i-p+j,i}/R_{i-p+j,i-p+j}, \quad \begin{matrix} i=p,p+1,\ldots N, \\ k=i-p+2,\ldots i, \end{matrix} \\
&\ell_{k,i} = \begin{cases} c_i - \lambda & \text{if } i=k \\ b_i & \text{if } i=k+1 \\ 0 & \text{otherwise} \end{cases} \\
&R_{i,j} = 0 \text{ otherwise} \ .
\end{aligned}\right\} \quad (5.2.20)$$

The elements $R_{i,j}$ can now be relabelled to facilitate economy of storage in the computer memory. The matrix given in (5.2.10) becomes,

$$\begin{bmatrix}
R_{1,1}, R_{2,2}, & & & R_{p,p} & & 0 \\
& R_{2,1} & & R_{p,p-1} & & \\
& & \ddots & \vdots & \ddots & R_{N,p} \\
& & & R_{p,2} & & \\
& 0 & & R_{p,1} & & R_{N,2} \\
& & & & & R_{N,1}
\end{bmatrix}, \quad (5.2.21)$$

where $R_{i,j}$ becomes $R_{j,j-i+1}$,

and the associated recursive sequence now becomes,

$$\left.\begin{aligned}
&R_{1,1} = c_1 - \lambda \ , \\
&R_{i,2} = b_i \ , \quad R_{i,1} = c_i - \lambda - R^2_{i,2}/R_{i-1}, \quad i=2,3,\ldots p-1, \\
&R_{i,p} = d_i \ , \qquad\qquad i=p,p+1,\ldots n,
\end{aligned}\right\} \quad (5.2.22)$$

$$R_{i,j} = 0 \quad , \qquad i<p \text{ and } i>2 \text{ or, } \quad i<j$$

$$R_{i,k} = \ell_k - \sum_{j=0}^{p-k-1} R_{i,p-j} R_{i-k+1,p+1-k-j} / R_{i-p+j+1,1}, \quad i=p,p+1,\ldots N,$$
$$k=1,2,\ldots.p-1,$$

$$\ell_k = \begin{cases} c_i - \lambda & \text{if } k=1 \\ b_i & \text{if } k=2 \\ 0 & \text{otherwise} \end{cases}$$

(5.2.22)

The sequence of elements $R_{i,j}$ can now be obtained from the relationship given in (5.2.22), and Gaussian elimination need not be referred to when calculating these new or reduced elements.

In the algorithm given as above in (5.2.22) no space is reserved for the elements $R_{i,j}$ for values of i<j, and in all the calculations involving these values of $R_{i,j}$ the procedure is to intercept them by testing for i<j and branch without carrying them out thus saving time.

If the elements $R_{i,1}$ and $R_{i,2}$, i=1,.....N are now arranged to be stored as two separate vectors, the total storage space needed for the sequence is $Np-p^2+3p-3$ computer words, (if each $R_{i,j}$ can be stored in a computer word)...

Since N and p are both often large with N>>p then the storage required is approximately of the order Np words. If N is approximately equal to p, then the storage required is approximately of the order of 3N words.

The repeated divisions by the elements of the array $R_{i,i}$ initially appears to be a highly unstable and dangerous practice if any element $R_{i,i}$ becomes zero or nearly zero. However this is dealt with in a similar manner to the procedure adopted by Barth et al (1967) i.e. by replacing the zero element $R_{i,i}$ by a small quantity close to machine zero (i.e. 2^{-k+1}, where k is the number of binary digits in the mantissa).

If $P_i(\lambda)$ i=0,N are the leading principal minors of the matrix (C-λI) then from Martin and Wilkinson (1967) there is the result which can be

written in terms of the current notation in (5.2.21) as,

$$P_{i+1}(\lambda) = (-1)^{s_{i+1}} R_{1,1},R_{2,1},R_{3,1},\ldots\ldots,R_{i+1,1} \quad , \tag{5.2.23}$$

where s_{i+1} is the number of interchanges. For this algorithm s_{i+1} is always zero and the term $(-1)^{s_{i+1}}$ can be omitted giving,

$$P_{i+1}(\lambda) = R_{1,1},R_{2,1},\ldots\ldots,R_{i+1,1} \quad , \tag{5.2.24}$$

and for the i^{th} principal minor,

$$P_{i}(\lambda) = R_{1,1},R_{2,1},\ldots\ldots\ldots,R_{i,1} \quad . \tag{5.2.25}$$

Combining (5.2.24) and (5.2.25) gives,

$$\frac{P_{i+1}(\lambda)}{P_{i}(\lambda)} = R_{i+1,1} \tag{5.2.26}$$

Since the polynomials $P_i(\lambda)$ i=0,1,2,...,N form a sequence of the leading principal minors of $|C-\lambda I|$ where C is a symmetric matrix, then it is well known that they form a properly signed interleaved sequence of polynomials (i.e., all $P_j(\lambda)>0$ for a sufficiently large value of either positive or negative and the zeros of $P_j(\lambda)$ strictly separate those of $P_{j+1}(\lambda)$). Therefore with the aid of the separation theorem Wilkinson (1965), it can be shown that the sequence $P_i(\lambda)$, i=0,1,...N form a Sturm sequence of polynomials in the interval $(-\infty,+\infty)$. The fundamental property of such polynomials facilitate an easy and simple method for the calculation of the roots by the process of bisection, i.e., the number of disagreements in sign in the sequence $P_j(\lambda)$, j=0,1,2,...N is equal to the number of roots of $P_N(\lambda)$ smaller than λ. Since the elements $R_{i,1}$, i=1,2,...N are ratios of the polynomials $P_i(\lambda)$ as given in (5.2.26) then it can be shown that the number of negative values of the elements $R_{i,1}$, i=1,2,...,N equals the number of roots of $P_N(\lambda)$ smaller than λ.

From Gerschgorin's theorem it is known that the eigenvalues of the matrix C (5.2.1) are all contained in the union of the N intervals,

$$c_i \pm (|b_i|+|b_{i+1}|+|d_{i+p-1}|+|d_i|) \ , \ i=1,2,\ldots,N \quad (5.2.27)$$

with $b_1 = b_{N+1} = 0$, $d_i = 0$, $i<p$, $i>N$.

Hence upper and lower bounds for the bisection process can be derived and are given by the expression,

$$\max_i \min_i (c_i \pm (|b_i|+|b_{i+1}|+|d_{i+p-1}|+|d_i|)) \quad (5.2.28)$$

The order in which the sequences of $R_{I,J}$ are obtained suggests a modification to the bisection algorithm which will in many cases ensure savings in time over the method used by Barth et al (1967). The sequence $R_{I,J}$ is obtained in a strict column order whereas in a Gaussian elimination process work is done on all remaining rows at each stage, and the elements in the pivotal row are found at the same time.

In the algorithm described by Barth et al (1967) a trial value λ (say) is chosen, and the whole Sturm sequence is calculated using this value. If k elements of the sequence are negative then λ is now used as a lower bound for the N-k+1 largest eigenvalues, and an upper bound for the k smallest eigenvalues. The bisection for this particular eigenvalue is then continued. As each eigenvalue is found, sharper and sharper bounds are obtained for the remaining eigenvalues. This reduces the number of bisections necessary to determine later eigenvalues, and makes full use of the information available.

As each member of the sequence $R_{I,1}$, I=1,N, is obtained before any work is done towards finding $R_{I+1,1}$, the signs of the $R_{I,1}$ sequence can be inspected as they are found. If a decision to bisect can be made at this point the sequence can be recomputed with a new value of λ without any loss of efficiency.

If, for instance, the smallest eigenvalue is being sought, as soon as a negative $R_{I,1}$, I=1,N is obtained the calculation of the sequence can be

stopped. It is now known that the value of λ chosen is too large, and a smaller one can be chosen. Calculation of the sequence of R's with the new value of λ can now be started, and the process repeated. Also if, for instance, a larger eigenvalue is being sought (J^{th} say), and the sequence has been calculated as far as $R_{I,1}$ with k members of the sequence $R_{L,1}$,L=1,I being negative. If N-I+k is less than J then the value of λ used is too small. (Less than J eigenvalues below λ therefore J^{th} has greater value than λ). Calculation of the R's sequence can be stopped and restarted with a larger value of λ.

This method has several advantages. All the computational effort is involved is solely in determining the particular eigenvalue being sought. No effort is expended finding bounds of other eigenvalues. If only a few eigenvalues are being sought (J say, where J<<N) this method is the most efficient. For a matrix of semi-bandwidth p the first p-1 members of the sequence $R_{I,1}$, I=1,N are trivial to compute compared to the remaining N-P+1 members of the sequence. If, therefore, a decision to stop calculation of the sequence and bisect for a new value of λ can be made before the calculation of the $R_{P,1}$ begin, great savings in time can be made.

In practice this algorithm (Program 12A in Appendix 1) is faster than that of Barth et al (1967) when only a few eigenvalues are sought. Where all eigenvalues are to be obtained in all examples tested there have been small differences in time between the two algorithms.

TYPE OF MATRIX	PROGRAM 12A	ALGORITHM OF BARTH ET AL.
20×20 matrix p=11 10 eigenvalues	15 seconds	62 seconds
25×25 matrix p=15 10 eigenvalues	48 seconds	59 seconds
30×30 matrix p=22 30 eigenvalues	233 seconds	272 seconds

TABLE 5.2.1

EXAMPLES OF RUN TIMES FOR TWO BISECTION ALGORITHMS

Tewarson (1973) has suggested some improvements in determining the sequence itself. The drop tolerance is a small value (Tewarson indicates from experience 10^{-7} is the best value) such that if an off-diagonal element's ($R_{I,J}$ where $J \neq 1$) modulus falls below this the element is replaced by zero. The pivot tolerance is a minimum value for a pivot ($R_{I,1}$, I=1,N) If a pivot has a modulus less than the pivot tolerance it is replaced. (Tewarson indicates a pivot tolerance of 10^{-3} when 9-10 figures of accuracy are being sought).

These improvements were included in the algorithm. The biggest effect was when multiple eigenvalues occurred, the accuracy of these were always improved. Of course all the tests involved increased the time taken by the algorithm.

5.3 CALCULATION OF THE EIGENVECTORS BY INVERSE ITERATION

Suppose $\underline{x}$ is taken as an initial trial vector, then to find the eigenvector corresponding to an eigenvalue λ (say) derived from (5.2.22) using two steps of an inverse iteration process the following two equations must be solved:-

$$(C-\lambda I)\underline{x} = \underline{y} \qquad (5.3.1)$$

and

$$(C-\lambda I)\underline{z} = \underline{x} \qquad (5.3.2)$$

where C is the quindiagonal matrix as given in (5.2.1).

To enable this to be carried out requires a knowledge of the inverse of the matrix $(C-\lambda I)$. This can be factorised and written in the form,

$$(C-\lambda I) = LU \qquad (5.3.3)$$

where L is a unit lower triangular matrix, and U is an upper triangular matrix as given by the triangular decomposition process on the matrix $(C-\lambda I)$.

If L and U are known then equations (5.3.1) and (5.3.2) can be solved by a forward and backward substitution process on the right hand side vector y. The matrices L and U can be determined by a Gaussian elimination process but in order to ensure numerical stability it is essential that pivoting techniques be incorporated in the solution process.

The matrix can be split into several arrays, to speed up computation, and save storage space in the computer. This is carried out as follows:-

$$(C-\lambda I)= \begin{bmatrix} F_1 & G_2 & H_3 & & & & & Q_{1,p-1} & \cdots & Q_{1,N} \\ E_1 & F_2 & G_3 & H_4 & & 0 & & Q_{2,p-1} & \cdots & Q_{2,N} \\ & \ddots & \ddots & \ddots & \ddots & & & \vdots & & \vdots \\ & & & & & & H_{p-2} & & & \\ & & & 0 & & & G_{p-2} & & & \\ & & & & & & F_{p-2} & & & \\ & & & & & & E_{p-1} & & & \\ A_{p,1} & \cdots & \cdots & \cdots & \cdots & \cdots & A_{p,p-2} & & & \\ A_{p+1,1} & & & & & & \vdots & & & \\ \vdots & & & & & & & & & \\ A_{N,1} & \cdots & \cdots & \cdots & \cdots & \cdots & A_{N,p-2} & Q_{N,p-1} & \cdots & Q_{N,N} \end{bmatrix}$$

(5.3.4)

Now the elimination process takes place in three major stages.

STAGE 1 The variables x_i, i=1,2,....p-4 are eliminated by using the following strategy.

At the j^{th} stage (say), the row with the maximum modulus of coefficient of the x_j is selected as the pivot element. Here there are N-p+3 rows at most to choose from. The j^{th} row (will be),

$$F_j x_j + G_{j+1} x_{j+1} + H_{j+2} x_{j+2} + \sum_{k=p-1}^{N} Q_{j,k} x_k \ , \qquad (5.3.5)$$

the $j+1^{th}$ row is

$$E_{j+1} x_j + F_{j+1} x_{j+1} + G_{j+2} x_{j+2} + \sum_{k=p-1}^{N} Q_{j+1,k} x_k \ , \qquad (5.3.6)$$

and the p to N^{th} rows are denoted by

$$\left.\begin{aligned} & A_{m,j} x_j + A_{m,j+1} x_{j+1} + A_{m,j+2} x_{j+2} + \sum_{k=p-1}^{N} Q_{m,k} x_k \ , m=p,N \\ & A_{m,j} = 0 \qquad m=p,p+1,\ldots N \qquad j=j+3,j+4,\ldots p-2 \ . \end{aligned}\right\} \qquad (5.3.7)$$

Thus if E_{j+1} has a maximum modulus the elements in (5.3.5) and (5.3.6) are interchanged as follows:-

$$\left.\begin{aligned} & F_j \leftrightarrow E_{j+1}, \quad G_{j+1} \leftrightarrow F_{j+1}, \quad H_{j+2} \leftrightarrow G_{j+2} \\ & Q_{j,k} \leftrightarrow Q_{j+1,k}, \quad k=p-1,p-2,\ldots,N, \end{aligned}\right\} \quad (5.3.8)$$

and if $U_{\ell,j}$, ℓ=N-p+1,N (say) has maximum modulus the elements in (5.3.5) and (5.3.7) are interchanged, i.e.,

$$\left.\begin{aligned} & F_j \leftrightarrow A_{\ell,j}, \quad G_{j+1} \leftrightarrow A_{\ell,j+1}, \quad H_{j+2} \leftrightarrow A_{\ell,j+2}, \\ & Q_{j,k} \leftrightarrow Q_{\ell,k}, \quad k=p-1,N \end{aligned}\right\} \quad (5.3.9)$$

The j^{th} element of the interchange vector is now set to zero for no interchange or to the number of the row with which row j was interchanged.

To eliminate the $j+1^{th}$ row E_j is set to E_{j+1}/F_j, then the following calculations are performed

$$\left.\begin{aligned} & F_{j+1} = F_{j+1} - G_{j+1}E_j, \quad G_{j+2} = G_{j+2} - H_{j+2}E_j, \\ & Q_{j+1,k} = Q_{j+1,k} - Q_{j,k}E_j, \quad k=p-1,N, \end{aligned}\right\} \quad (5.3.10)$$

and the N-p+1 rows of A become,

$$\left.\begin{aligned} & A_{m,j} = A_{m,j}/F_j \qquad m=p,N \\ & A_{m,j+1} = A_{m,j+1} - G_{j+1}A_{m,j}, \quad A_{m,j+2} = A_{m,j+2} - Z_{j+2}A_{m,j} \\ & A_{m,k} = A_{m,k} - Q_{j,k}A_{m,j}, \quad k=p-1,N \end{aligned}\right\} \begin{aligned} & m=p,N \\ & (5.3.11) \end{aligned}$$

STAGE 2 The variables x_{p-3}, x_{p-2} are now eliminated. This stage is essentially the same as the first stage, except that there are fewer terms outside the Q submatrix.

For the variable x_{p-3}, j=p-3 and (5.3.5), (5.3.6) and (5.3.7) become,

$$\left.\begin{aligned} & F_{p-3}x_{p-3} + G_{p-2}x_{p-2} + \sum_{k=p-1}^{N} Q_{p-3,k}x_k, \\ & E_{p-2}x_{p-3} + F_{p-2}x_{p-2} + \sum_{k=p-1}^{N} Q_{p-2,k}x_k, \\ & A_{m,p-3}x_{p-3} + A_{m,p-2}x_{p-2} + \sum_{k=p-1}^{N} Q_{m,k}x_k, \quad m=p,N, \end{aligned}\right\} \quad (5.3.12)$$

the terms involving H_{p-1}, G_{p-1}, $A_{m,p-1}$, m=p,N are now in the sub-matrix Q and need not be considered separately.

If E_{p-2} is the coefficient of x_{p-3} with maximum modulus the following elements are interchanged,

$$F_{p-3} \leftrightarrow E_{p-2},\quad G_{p-2} \leftrightarrow F_{p-2},\quad Q_{p-3,k} \leftrightarrow Q_{p-2,k},\ k=p-1,N, \tag{5.3.13}$$

and if $H_{\ell,p-3}$ (say) is the coefficient of maximum modulus the following elements are interchanged,

$$F_{p-3} \leftrightarrow A_{\ell,p-3},\ G_{p-2} \leftrightarrow A_{\ell,p-2},\ Q_{p-3,k} \leftrightarrow Q_{\ell,k},\ k=p-1,N\ . \tag{5.3.14}$$

The interchange (or no interchange) is again noted in the interchange vector. Now x_{p-3} is eliminated in the $p-2^{th}$ row as follows,

$$\left.\begin{aligned} &E_{p-3} = E_{p-2}/F_{p-3},\quad F_{p-2} = F_{p-2} - G_{p-2}E_{p-3}\ , \\ &Q_{p-2,k} = Q_{p-2,k} - Q_{p-3,k}\,G_{p-3},\ k=p-1,N, \end{aligned}\right\} \tag{5.3.15}$$

and for the elimination in the N-p+1 rows of A,

$$\left.\begin{aligned} &A_{m,p-3} = A_{m,p-3}/F_{p-3}\ , \\ &A_{m,p-2} = A_{m,p-2} - G_{p-2}A_{m,p-3}\ , \\ &Q_{m,k} = Q_{m,k} - Q_{p-3,k}A_{m,p-3},\ k=p-1,N \end{aligned}\right\} \ m=p,N \tag{5.3.16}$$

For the variable x_{p-2}, j=p-2 and (5.3.5), (5.3.6), and (5.3.7) become,

$$\left.\begin{aligned} &F_{p-2}x_{p-2} + \sum_{k=p-1}^{N} Q_{p-2,k}x_k\ , \\ &E_{p-1}x_{p-2} + \sum_{k=p-1}^{N} Q_{p-1,k}x_k\ , \\ &A_{m,p-2}x_{p-2} + \sum_{k=p-1}^{N} Q_{m,k}x_k,\ m=p,N\ , \end{aligned}\right\} \tag{5.3.17}$$

the terms involving H_p, G_{p-1}, G_p and F_{p-1} are now in the sub-matrix Q and need not be considered separately.

If E_{p-1} is the coefficient of x_{p-2} with maximum modulus the following elements are interchanged,

$$F_{p-2} \leftrightarrow E_{p-1},\quad Q_{p-2,k} \leftrightarrow Q_{p-1,k},\quad k=p-1,N\ , \tag{5.3.18}$$

and if $A_{\ell,p-3}$(say) is the coefficient of maximum modulus the following elements are interchanged,

$$F_{p-2} \leftrightarrow A_{\ell,p-2}, \quad Q_{p-2,k} \leftrightarrow Q_{\ell,k}, \quad k=p-1,N, \tag{5.3.19}$$

and the interchange noted in the interchange vector.

Now x_{p-2} is eliminated in the $p-1^{th}$ row as follows,

$$\left.\begin{aligned} E_{p-2} &= E_{p-1}/F_{p-2}, \\ Q_{p-1,k} &= Q_{p-1,k} - Q_{p-2,k}E_{p-2}, \quad k=p-1,N, \end{aligned}\right\} \tag{5.3.20}$$

and for the elimination in the N-p+1 rows of A,

$$\left.\begin{aligned} A_{m,p-2} &= A_{m,p-2}/F_{p-2}, \\ Q_{m,k} &= Q_{m,k} - Q_{p-2,k}A_{m,p-2}, \quad k=p-1,N \end{aligned}\right\} \quad m=p,N \tag{5.3.21}$$

STAGE 3 The variables x_i, i=p-1,N-1 are now eliminated. The rows of original matrix $(C-\lambda I)$ with x_i, i=p-1,N-1 still to be eliminated are now wholly contained in the lower p-1 rows of the sub-matrix Q. These rows form a $(p-1)\times(p-1)$ matrix upon which the standard Gaussian elimination procedure with partial pivoting strategies are performed. The elimination factors are stored in the lower triangle of Q below the diagonal and interchanges are noted in the interchange vector.

The information now stored in A,E,F,G,H,Q now provides sufficient information to solve the equations,

$$(C-\lambda I)\underline{x} = \underline{y}$$

for any right hand side vector y by the appropriate forward and backward substitutions. Hence (5.3.1) can be written in the form

$$LU\underline{x} = \underline{y} \tag{5.3.22}$$

provided the interchanges in L are included.

Wilkinson (1965) has shown that if the initial vector $\underline{y}$ is of the form,

$$\underline{y} = L\,\underline{e}, \tag{5.3.23}$$

where the vector $\underline{e}^T$ is of the form (1,1,...1), then, substituting (5.3.23) in (5.3.22) gives the result,

$$U\underline{x} = \underline{e} \quad , \tag{5.3.24}$$

where the upper triangular matrix U has the general form:-

$$U = \begin{bmatrix} F_1 & G_2 & H_3 & & & 0 & Q_{1,p-1} & \cdots & Q_{1,N} \\ & \ddots & \ddots & \ddots & & & Q_{2,p-1} & \cdots & \\ & & & & H_{p-2} & & \vdots & & \vdots \\ & & & & G_{p-2} & & & & \\ & & & & F_{p-2} & & Q_{p-2,p-1} & & \\ & & & & & & Q_{p-1,p-1} & \ddots & \\ & & 0 & & & & & \ddots & Q_{N,N} \end{bmatrix} \tag{5.3.25}$$

With this choice of $\underline{y}$, $\underline{x}$ can be determined using a simple back-substitution given in algorithmic form as:

$$\left.\begin{aligned} & x_n = y_n/Q_{n,n} \quad , \\ & \text{for } i = n-1(-1)p-1, \\ & x_i = (y_i - \sum_{j=i-1}^{N} Q_{i,j}x_j)/Q_{ii} \quad , \\ & \text{for } i = p-2(-1)1, \\ & x_i = (y_i - \sum_{j=p-1}^{N} Q_{i,j}x_j - Q_{i+1}x_{i+1} - H_{i+2}x_{i+2})/F_i, \\ & H_{p-1} = H_p = G_{p-1} = 0 \quad . \end{aligned}\right\} \tag{5.3.26}$$

Thus there is no need to determine $L\underline{e}$ explicitly. To perform the forward substitution operating on the x_i, the stored elimination factors in E,A, and Q, and the interchange vector IC (IC_i, i=1,N-1) are used in the following manner,

$$\left.\begin{array}{l}(i=1(1)p-2,\ (\text{if } IC_i\neq 0,\quad x_i \leftrightarrow x_{IC_i})\ ,\\ \qquad x_{i+1} = x_{i+1} - x_i E_{i-1}\ ,\\ \quad (m=p(1)N,\ \ x_m = x_m - x_i A_{m,i}))\ ,\\ (i=p-1(1)N,\ \ (\text{if } IC_i\neq 0,\quad x_i \leftrightarrow x_{IC_i})\ ,\\ \quad (m=i(1)N\ \ x_m = x_m - x_i Q_{m,i}))\,.\end{array}\right\} \qquad (5.3.27)$$

If the back substitution process described in (5.3.26) is performed the required eigenvector is now in $\underline{x}$.

The number of storage elements s, required for the arrays A,E,F,G,H, and Q is given by

$$N^2-p^2+7p-13 = s \qquad (5.3.28)$$

If N and p are very large then the approximate storage required is,

$$N^2-p^2 \simeq s. \qquad (5.3.29)$$

The quantity s can easily be seen to be small with large band-width matrices.

5.4 RESULTS

Results are given for two test matrices of the same form as (5.2.1c). The first is a (14×14) matrix with semi bandwidth 8, and elements as those of (5.2.1c) divided by 4. The eigenvalues of this matrix are given by the formula,

$$\lambda_{i,j} = 1 - \frac{1}{4}\cos\left(\frac{i\pi}{3}\right) - \frac{1}{4}\cos\left(\frac{j\pi}{8}\right) \quad i=1,2,\ j=1,2,\ldots 7 \qquad (5.4.1)$$

The answers given in table (5.4.1) agreed with those given by equation (5.4.1) to 10 significant figures. Each column of table (5.4.1) contains the eigenvalue at the head of its corresponding eigenvector. The eigen-vectors also agreed to 10 significant figures with those obtained by the N.A.G. routine FO2ABA which uses Householders reduction and the QL algorithm to obtain the eigenvalues and eigenvectors of a symmetric matrix.

0.2880602338	0.3964466094	0.5586582838	0.7499999999	0.7880602337	0.8964466095	0.9413417162
0.1352990250	-0.2500000000	0.3266407412	-0.3535533906	-0.1352990250	0.2500000000	0.3266407412
0.2500000000	-0.3535533906	0.2500000000	0.0000000000	-0.2500000000	0.3535533906	-0.2500000000
0.3266407412	-0.2500000000	-0.1352990250	0.3535533906	-0.3266407412	0.2500000000	-0.1352990250
0.3535533906	0.0000000000	-0.3535533906	0.0000000000	-0.3535533906	0.0000000000	0.3535533906
0.3266407412	0.2500000000	-0.1352990250	-0.3535533906	-0.3266407412	-0.2500000000	-0.1352990250
0.2500000000	0.3535533906	0.2500000000	0.0000000000	-0.2500000000	-0.3535533906	-0.2500000000
0.1352990250	0.2500000000	0.3266407412	0.3535533906	-0.1352990250	-0.2500000000	0.3266407412
0.1352990250	-0.2499999999	0.3266407412	-0.3535533906	0.1352990250	-0.2500000000	0.3266407412
0.2500000000	-0.3535533906	0.2500000000	0.0000000000	0.2500000000	-0.3535533906	-0.2500000000
0.3266407412	-0.2500000000	-0.1352990250	0.3535533906	0.3266407412	-0.2500000000	-0.1352990250
0.3535533906	0.0000000000	-0.3535533906	0.0000000000	0.3535533906	0.0000000000	0.3535533906
0.3266407412	0.2500000000	-0.1352990250	-0.3535533906	0.3266407412	0.2500000000	-0.1352990250
0.2500000000	0.3535533906	0.2500000000	0.0000000000	0.2500000000	0.3535533906	-0.2500000000
0.1352990250	0.2500000000	0.3266407412	0.3535533906	0.1352990250	0.2500000000	0.3266407412

1.058658284	1.103553391	1.211939766	1.250000000	1.441341717	1.603553391	1.7119397466
-0.3266407412	0.2500000000	0.1352990250	-0.3535533906	-0.3266407412	-0.2500000000	-0.1352990250
-0.2500000000	-0.3535533906	-0.2500000000	0.0000000000	0.2500000000	0.3535533906	0.2500000000
0.1352990250	0.2500000000	0 3266407412	0.3535533906	0.1352990250	-0.2500000000	-0.3266407412
0.3535533906	0.0000000000	-0.3535533906	0.0000000000	-0.3535533906	0.0000000000	0.3535533906
0.1352990250	-0.2500000000	0.3266407412	-0.3535533906	0.1352990250	0.2500000000	0.3266407412
-0.2500000000	0.3535533906	-0.2500000000	0.0000000000	0.2500000000	-0.3535533906	0.2500000000
-0.3266407412	-0.2500000000	0.1352990250	0.3535533906	-0.3266407412	0.2500000000	-0.1352990250
0.3266407412	0.2500000000	0.1352990250	0.3535533906	0.3266407412	0.2500000000	0.1352990250
0.2500000000	-0.3535533906	-0.2500000000	0.0000000000	-0.2500000000	-0.3535533906	-0.2500000000
-0.1352990250	0.2500000000	0.3266407142	-0.3535533906	-0.1352990250	0.2500000000	0.3266407412
-0.3535533906	0.0000000000	-0.3535533906	0.0000000000	0.3535533906	0.0000000000	-0.3535533906
-0.1352990250	-0.2500000000	0.3266407412	0.3535533906	-0.1352990250	0.2500000000	0.3266407412
0.2500000000	0.3535533906	-0.2500000000	0.0000000000	-0.2500000000	0.3535533906	-0.2500000000
0.3266407412	-0.2500000000	0.1352990250	-0.3535533906	0.3266407412	-0.2500000000	0.1352990250

TABLE 5.4.1

The second matrix used is of the same form as the first, but is of order (80×80) with semi-bandwidth 41. Again the eigenvalues of this matrix are well known, and are given by,

$$\lambda_{i,j} = 1 - \frac{1}{4}\cos\left(\frac{i\pi}{3}\right) - \frac{1}{4}\cos\left(\frac{j\pi}{41}\right),\ i=1,2,\ j=1,2,\ldots 40 \quad (5.4.2)$$

For this matrix eigenvalues 1,11,21,31,41,51,61,71 are given in table (5.4.2) where eigenvalue 1 is the smallest. The results again agreed to 10 significant figures with those obtained using (5.4.2). The eigenvector associated with eigenvalue 1 is given in table (5.4.3) and the results agreed to 10 significant figures with those obtained using the QL algorithm as before.

No.	EIGENVALUE
1	0.2514670994
11	0.4173371499
21	0.7514670994
31	0.8451955545
41	1.011140091
51	1.158964680
61	1.269151367
71	1.610260797

TABLE 5.4.2

x_1-x_{20}	x_{21}-x_{40}	x_{41}-x_{60}	x_{61}-x_{80}
$1.195498477*10^{-2}$	$1.560591587*10^{-1}$	$1.1954984770*10^{-2}$	$1.560591586*10^{-1}$
$2.383981293*10^{-2}$	$1.551433415*10^{-1}$	$2.383981294*10^{-2}$	$1.551433415*10^{-1}$
$3.558473960*10^{-2}$	$1.5331708145*10^{-1}$	$3.558473960*10^{-2}$	$1.533170814*10^{-1}$
$4.712084086*10^{-2}$	$1.505910959*10^{-1}$	$4.712084086*10^{-2}$	$1.505910959*10^{-1}$
$5.838041829*10^{-2}$	$1.469813818*10^{-1}$	$5.838041829*10^{-2}$	$1.469813818*10^{-1}$
$6.929739622*10^{-2}$	$1.425091226*10^{-1}$	$6.929739622*10^{-2}$	$1.425091226*10^{-1}$
$7.980770947*10^{-2}$	$1.372005632*10^{-1}$	$7.980770947*10^{-2}$	$1.372005632*10^{-1}$
$8.984967935*10^{-2}$	$1.310868563*10^{-1}$	$8.984967950*10^{-2}$	$1.310868563*10^{-1}$
$9.936437559*10^{-2}$	$1.242038796*10^{-1}$	$9.936437559*10^{-2}$	$1.242038796*10^{-1}$
$1.082959622*10^{-1}$	$1.165920251*10^{-1}$	$1.082959622*10^{-2}$	$1.165920251*10^{-1}$
$1.165920250*10^{-1}$	$1.082959623*10^{-1}$	$1.165920250*10^{-1}$	$1.082959623*10^{-1}$
$1.242038794*10^{-1}$	$9.936437575*10^{-2}$	$1.242038794*10^{-1}$	$9.936437575*10^{-2}$
$1.310868561*10^{-1}$	$8.984967950*10^{-2}$	$1.310868562*10^{-1}$	$8.984967950*10^{-2}$
$1.372005630*10^{-1}$	$7.980770961*10^{-2}$	$1.372005630*10^{-1}$	$7.980770961*10^{-2}$
$1.425091225*10^{-1}$	$6.929739634*10^{-2}$	$1.425091225*10^{-1}$	$6.929739634*10^{-2}$
$1.469813817*10^{-1}$	$5.838041840*10^{-2}$	$1.469813817*10^{-1}$	$5.838041840*10^{-2}$
$1.505910958*10^{-1}$	$4.712084094*10^{-2}$	$1.505910958*10^{-1}$	$4.712084094*10^{-2}$
$1.533170814*10^{-1}$	$3.558473966*10^{-2}$	$1.533170814*10^{-1}$	$3.558473966*10^{-2}$
$1.551433414*10^{-1}$	$2.383981298*10^{-2}$	$1.551433414*10^{-1}$	$2.383981298*10^{-2}$
$1.560591587*10^{-1}$	$1.195498479*10^{-2}$	$1.560591586*10^{-1}$	$1.195498479*10^{-2}$

TABLE 5.4.3

5.5 DETERMINATION OF THE STURM SEQUENCE FOR AN UNSYMMETRIC BANDED MATRIX AND ITS USE IN FINDING EIGENVALUES

The method of obtaining the Sturm sequence for a matrix as explained in 5.2 can be extended and used on the more difficult unsymmetric matrix of the same form as (5.2.1). As the matrix is unsymmetric, the Sturm sequence, once found, cannot be used in a bisection process as any number of the eigenvalues may be complex. Instead a root finding method normally associated with polynomial root finding is used. The Sturm sequence is now utilised to obtain the determinant only of the matrix which can be used in conjunction with the root finding method to find the roots of the characteristic equation. Obtaining a suitable differentiated sequence as described in Chapter 3 for use with Newton's method is impracticable, so Muller's method is used even though it has several drawbacks. At each stage the method requires two previous function (determinant) evaluations. Thus choosing two suitable starting values is a problem, and often a cause of inefficiencies. The method can, and often does, give a complex approximation to a real root, but this is in common with many other methods. There is one square root evaluation at every iteration, which is time consuming. In fact computationally the single most time consuming operation in the method is the determination of the square root of a complex number, but this is very small compared to the amount of work performed in one iteration.

The matrix for which the sequence is to be found is

$$A = \begin{bmatrix} c_1 & b_2 & & & d_p & & 0 \\ e_2 & c_2 & \ddots & & & \ddots & \\ & \ddots & \ddots & \ddots & 0 & & d_N \\ & & \ddots & \ddots & \ddots & \ddots & \\ f_p & & 0 & \ddots & \ddots & \ddots & b_N \\ & \ddots & & \ddots & \ddots & \ddots & \\ 0 & & \ddots & & e_N & c_N & \end{bmatrix} \qquad (5.5.1)$$

As before Gaussian elimination is applied to the matrix $(A-\lambda I)$ without using a pivoting strategy. At each stage of the elimination process some of the elements will be relabelled, and initially the following are relabelled:

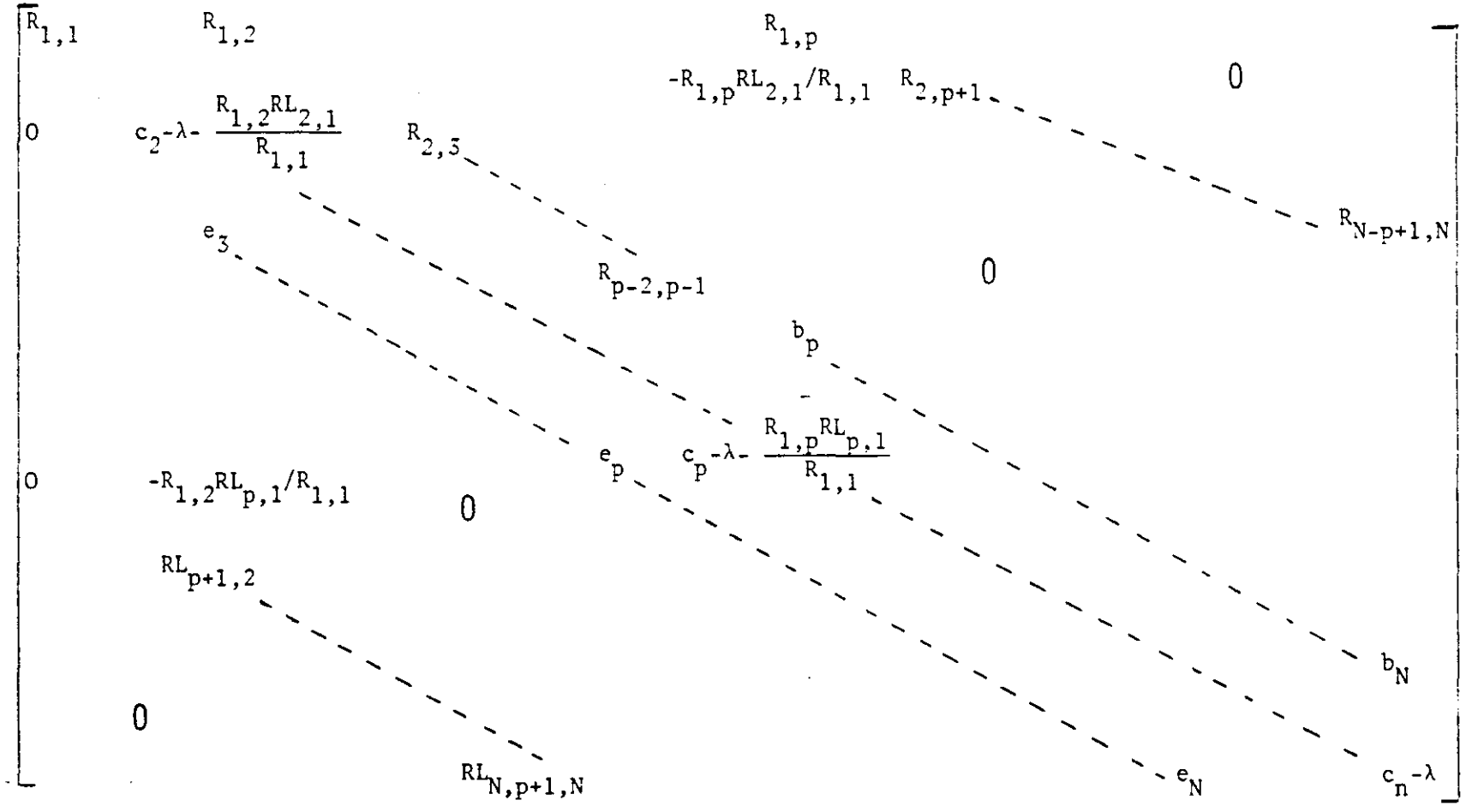
$$\left.\begin{aligned} R_{i,p-1+i} &= d_{i+p-1} \quad , \quad i=1,2,\ldots,N-p+1 \\ RL_{p+i-1,i} &= f_{i+p-1} \quad , \quad i=1,2,\ldots,N-p+1 \end{aligned}\right\} \qquad (5.5.2)$$

$$\left.\begin{aligned} R_{i,i+1} &= b_i \quad , \quad i=2,3,\ldots,p-1 \\ R_{1,1} &= c_1-\lambda \quad , \quad e_2 = RL_{2,1} \, . \end{aligned}\right\} \qquad (5.5.3)$$

Thus before the first step of elimination the matrix now has the form,

$$(A-\lambda I) = \begin{bmatrix} R_{1,1} & R_{1,2} & & & R_{1,p} & & \\ RL_{2,1} & c_2-\lambda & R_{2,3} & & & R_{2,p+1} & 0 \\ & e_3 & \ddots & \ddots & & \ddots & \\ & & \ddots & \ddots & R_{p-2,p-1} & 0 & R_{N-p+1,N} \\ & & & \ddots & \ddots & \ddots & \\ & & & & b_p & \ddots & \\ RL_{p,1} & & & & \ddots & \ddots & \\ & RL_{p+1,2} & 0 & & & \ddots & b_N \\ & \ddots & \ddots & & & & \\ 0 & & \ddots & & & e_N & c_N-\lambda \end{bmatrix} \qquad (5.5.4)$$

and performing the first step of the elimination will produce,



(5.5.5)

Now some more elements can be relabelled as follows,

$$\left.\begin{aligned} R_{2,2} &= c_2-\lambda-R_{1,2}RL_{2,1}/R_{1,1} \quad , \\ RL_{3,2} &= e_3, \quad R_{2,p} = -R_{1,p}RL_{2,1}/R_{1,1} \quad , \\ RL_{p,2} &= -R_{1,2}RL_{p,1}/R_{1,1} \quad , \end{aligned}\right\} \qquad (5.5.6)$$

and the next step of the Gaussian elimination process is performed when the matrix becomes,

$$
\begin{bmatrix}
R_{1,1} & R_{1,2} & & & R_{1,p} & & \\
 & R_{2,2} & R_{2,3} & & R_{2,p} & R_{2,p+1} & 0 \\
 & & c_3-\lambda-R_{2,3}RL_{3,2}/R_{2,2} & R_{3,4} & -R_{2,p}RL_{3,2}/R_{2,2} & -R_{2,p+1}RL_{3,2}/R_{2,2} & R_{3,p+2} \\
 & & e_4 & \ddots & & & \ddots \\
 & & & \ddots & R_{p-2,p-1} & & R_{N-p+1,N} \\
 & & & & b_p & & 0 \\
 & & & & c_p-\lambda-\frac{R_{1,p}RL_{p,1}}{R_{1,1}} & b_{p+1} & \\
0 & 0 & -RL_{p,2}R_{2,3}/R_{2,2} & e_p & -R_{2,p}RL_{p,2}/R_{2,2} & R_{2,p+1}RL_{p,2}/R_{2,2} & \\
 & 0 & -RL_{2,p+1}R_{2,3}/R_{2,2} & & e_{p+1} & c_{p+1}-\lambda- & b_{p+2} \\
 & & & & RL_{p+1,2}R_{2,p}/R_{2,2} & -R_{2,p+1}RL_{p+1,2}/R_{2,2} & \\
 & & RL_{p+2,3} & 0 & & e_{p+2} & \ddots \\
 & 0 & & \ddots & & \ddots & b_N \\
 & & & RL_{N-p+1,N} & & e_N & c_N-\lambda
\end{bmatrix}
\tag{5.5.7}
$$

The process of eliminating RL elements and relabelling is now continued until an upper triangular matrix is left,

$$\begin{bmatrix} R_{1,1} & R_{1,2} & & & R_{1,p} & & 0 \\ & R_{2,2} & R_{2,3} & 0 & R_{2,p} & R_{2,p+1} & \\ & & R_{3,3} & & & R_{3,p+1} & \ddots \\ & & & \ddots & & & \ddots & R_{N-p+1,N} \\ & & & & R_{p-1,p} & & \\ & & & & R_{p,p} & \ddots & \\ & & 0 & & & \ddots & R_{N-1,N} \\ & & & & & & R_{N,N} \end{bmatrix} \qquad (5.5.8)$$

In this process a sequence of R elements have been produced and recorded, and a sequence of RL elements which are the last values of an element in that position in the matrix before elimination. These RL elements can all be relabelled so that elements in symmetric positions now have the same indices e.g.,

$$\left.\begin{array}{lll} RL_{2,1} & \text{becomes} & RL_{1,2} \\ RL_{p,1} & \text{becomes} & RL_{1,p} \end{array}\right\} \qquad (5.5.9)$$

The values of the $R_{i,j}$, $RL_{i,j}$ (i=p,N, j=p,N) can be determined by the same method as described in section 5.2, while the $R_{i,j}$, $RL_{i,j}$ (i=1,p, j=i,i+1) are determined in the same manner as a normal tridiagonal sequence.

A further, and final, relabelling of the elements $RL_{i,j}$ and $R_{i,j}$ can now be performed to maximise efficiency and minimise storage required when programmed on a computer. The matrix (5.5.8) now becomes,

$$
\begin{bmatrix}
R_{1,1} & R_{2,2} & & & R_{p,p} & & \\
 & R_{2,1} & R_{3,2} & 0 & R_{p,p-1} & R_{p+1,p} & 0 \\
 & & \ddots & \ddots & \vdots & R_{p+1,p-1} & \ddots \\
 & & & \ddots & R_{p,2} & \ddots & R_{N,p} \\
 & & & & R_{p,1} & & \\
 & & & 0 & & & \\
 & & & & & & R_{N,2} \\
 & & & & & & R_{N,1}
\end{bmatrix}
\tag{5.5.10}
$$

and the RL are relabelled similarly,

These R and RL elements can be obtained by Gaussian elimination and relabelling, or the following recursive formula,

$$
\left.
\begin{aligned}
&R_{i,k} = d_i, \qquad RL_{i,k} = f_i \quad , \ i=p,p+1,\ldots\ldots N, \\
&R_{i,k} = RL_{i,k} = 0 \quad i<p \ \text{ and } \ k>2, \\
&RL_{i,k} = \ell_{i,k} - \sum_{j=0}^{p-k-1} RL_{i,p-j} R_{i-k+1,p+1-k-j} / R_{i-p-j+1,1}, \\
&\qquad\qquad i=2,3,\ldots,N, \quad k=p,p-1,\ldots 2, \\
&R_{i,k} = \ell_{i,k} - \sum_{j=0}^{p-k-1} R_{i,p-j} RL_{i-k+1,p+1-k-j} / R_{i-p+j+1,1}, \\
&\qquad\qquad i=1,2,\ldots,N, \quad k=p,p-1,\ldots 2,
\end{aligned}
\right\}
\tag{5.5.11}
$$

$$
\ell_{i,k} = \begin{cases}
c_i - \lambda & \text{if} \quad k=1 \\
e_i & \text{if} \quad k=2 \text{ for } RL_{i,k} \\
b_i & \text{if} \quad k=2 \text{ for } R_{i,k} \\
0 & \text{otherwise.}
\end{cases}
$$

which was obtained from careful inspection of the elements during the elimination and relabelling process.

If $P_i(\lambda)$, i=1,N are the leading principal minors of the matrix (A-λI) and therefore $P_N(\lambda)$ is the determinant of (A-λI), then the $R_{i,1}(\lambda)$, i=1,N are the ratios of the $P_i(\lambda)$ i.e.,

$$\frac{P_i(\lambda)}{P_{i-1}(\lambda)} = R_{i,1}(\lambda) \qquad (5.5.12)$$

as shown by Martin (1967).

Therefore, the determinant of the matrix (A-λI) is given by,

$$P_N(\lambda) = R_{1,1}(\lambda)\ R_{2,1}(\lambda).....R_{N,1}(\lambda), \qquad (5.5.13)$$

and in general

$$P_i(\lambda) = R_{1,1}(\lambda)R_{2,1}(\lambda)....R_{i,1}, \quad i=1,2,...N, \qquad (5.5.14)$$

Muller's method is described well in Froberg (1964), but briefly the method fits a parabola to three previous function (determinant) evaluations, and uses this to determine an improved approximation to the root (eigenvalue). So that at the i^{th} step the previous evaluations are $(f_{i-2}(\lambda^{i-2}),\ f_{i-1}(\lambda^{i-1}),\ f_i(\lambda^i))$ where the λ's are approximations to an eigenvalue and $f_j(\lambda^j)=P_N(\lambda^j)$. Then if,

$$h_j = \lambda^j - \lambda^{j-1}, \quad \psi_j = h_j/h_{j-1}, \quad \delta_j = 1 + \psi_j\ , \qquad (5.5.15)$$

the following can be obtained,

$$g_i = f_{i-2}\psi_i^2 - f_{i-1}\delta_i^2 + f_i(\psi_i + \delta_i)\ , \qquad (5.5.16)$$

and then,

$$\psi_{i+1} = -2f_i\delta_i/(g_i \pm \sqrt{g_i^2 - 4f_i\delta_i\psi_i[f_{i-2}\psi_i - f_{i-1}\delta_i + f_i]}) \qquad (5.5.17)$$

Then by choosing the sign to make the modulus of the denominator as large as possible

$$\lambda^{i+1} = \lambda^i + \psi_{i+1}\ h_i\ . \qquad (5.5.18)$$

Now using (5.5.15-5.5.18) the iteration can be continued until convergence to an eigenvalue. In practice the form,

$$f_j(\lambda^j) = P_N(\lambda^j)\ , \qquad (5.5.19)$$

is not used when calculating the results on a computer. The reason is that outlined in Chapter 3, that the $P_i(\lambda)$ are subject to large oscillations and overflow occurs. As can be seen when calculating $P_N(\lambda^j)$ from (5.5.13) and (5.5.14) the quantities $P_i(\lambda^j)$, i=1,2,...N are all calculated, and are all subject to the possibility of overflow. If $P_N(\lambda^j)$ is calculated in the reverse manner then quantities,

$$\frac{P_N(\lambda^j)}{P_i(\lambda^j)} \quad , \; i=N-1,N-2,N-3,\ldots,0 \tag{5.5.20}$$

are all calculated and are again subject to overflow.

Instead of $P_N(\lambda^i)$, $R_{N,1}(\lambda^j)$ can be used, and then,

$$f_j(\lambda^j) = R_{N,1}(\lambda^j) \quad . \tag{5.5.21}$$

It was found that in practice using (5.5.21) did not affect the convergence in any way and satisfactory results were obtained without overflow occuring.

When one eigenvalue had been found it was necessary to deflate the matrix to prevent redetermination of this eigenvalue. This is achieved by dividing the function by the difference of the current estimate to an eigenvalue and all previously determined eigenvalues. Such that if i eigenvalues have already been found, then,

$$f_j(\lambda^j) = R_{N,1}(\lambda^j) / \prod_{k=1}^{i} (\lambda_k - \lambda^j) \tag{5.5.22}$$

Unfortunately this exact form cannot be used as it can be written,

$$f_j(\lambda^j) = P_N(\lambda^j) / \prod_{k=1}^{i} (\lambda_k - \lambda^j) P_{N-1}(\lambda^j) \quad . \tag{5.5.23}$$

This is in effect,

$$f_j(\lambda^j) = N^{th} \text{ degree polynomial}/(N-1+i)^{th} \text{ degree polynomial} \tag{5.5.24}$$

and $f_j(\lambda^j)$ has as an asymptote the line $f(\lambda)=0$ and the method always follows this asymptote, never converging. The solution to this problem

is a delicate "balancing act". As each eigenvalue is found the polynomial in the denominator of (5.5.22) is decreased by one and then $f_j(\lambda^{(j)})$ is given by,

$$f_j(\lambda^j) = \prod_{k=1}^{i+1} R_{N+1-k,1}(\lambda^j) / \prod_{\ell=1}^{i}(\lambda_\ell - \lambda^j) \qquad (5.5.25)$$

$$= P_N(\lambda^j)/P_{N-i-1}(\lambda^j) \prod_{k=1}^{i} (\lambda_k - \lambda^j) \qquad (5.5.26)$$

This gives, in effect, at every stage,

$$f_j(\lambda^j) = N^{th} \text{ degree polynomial}/(N-1)^{th} \text{ degree polynomial} \qquad (5.5.27)$$

and now the process converges at every step and overflow is avoided. The effect of using a ratio of the determinant to a minor instead of just the determinant is discussed in Chapter 7, in connection with the secant method. At this stage it can be said that in practice this had no effect on convergence or accuracy at all.

The program to perform this algorithm is given in the Appendix 1 in program 14.

5.6 RESULTS

Program 14 was run on a number of matrices which were chosen randomly and compared with results obtained from N.A.G. routine F02AJA. This routine reduces the matrix to upper Hessenberg form using stabilised similarity transformations, and then computes the eigenvalues using a modified LR method. Program 14 is written in Algol 60, this highlights how clumsy the language is in handling complex arithmetic. It was found that for the operations +,-,* it was quickest and most convenient to write them explicitly, and for the operations ÷, and square root they were written as subroutines. This was because they were used least to be the most complex, and also afforded the possibility of introducing scaling if necessary.

The first matrix used was, a 20×20 matrix with semi-bandwidth 14, and

random integers between 0-9 for the elements on the diagonals. The eigenvalues are listed, for comparison with those obtained using the N.A.G. library routine F02AJA. This routine uses stabilised elementary similarity transformations to transform the matrix to upper Hessenberg form, then uses the LR algorithm to obtain the eigenvalues. The two programs were run on the Loughborough University of Technology I.C.L. 1904S computer, and program 14 obtained the eigenvalues in 300 seconds, while F02AJA obtained the eigenvalues in 10 seconds. The results are given in table (5.6.1).

EIGENVALUES FROM PROGRAM 14		EIGENVALUES FROM F02AJA	
REAL	IMAGINARY	REAL	IMAGINARY
-11.64194747	0.000000000	-11.64194748	0.000000000
-2.240745232	0.000000000	-2.240745233	0.000000000
-0.0075399962	0.000000000	-0.0075399962	0.000000000
1.974288297	0.000000000	1.974288297	0.000000000
2.776414614	-2.222505556	2.776414614	-2.222505556
2.776414614	2.222505556	2.776414614	2.222505556
3.839793997	0.000000000	3.839793997	0.000000000
7.349843142	0.000000000	7.349843142	0.000000000
8.757966495	0.000000000	8.757966495	0.000000000
9.339058777	0.000000000	9.339059777	0.000000000
10.35427842	0.000000000	10.35427842	0.000000000
11.87654137	0.000000000	11.87654137	0.000000000
13.73292636	-3.391590422	13.73292636	-3.391590422
13.73292636	3.391590423	13.73292636	3.391590422
16.43962543	0.000000000	16.43962543	0.000000000
18.65614394	0.000000000	18.65614394	0.000000000
21.46577519	0.000000000	21.46577519	0.000000000
24.59456121	0.000000000	24.59456120	0.000000000
27.63531832	0.000000000	27.63531832	0.000000000
28.65621514	0.000000000	28.65621514	0.000000000

TABLE 5.6.1

A second matrix of order 50×50 with elements consisting of integers between 0 and 9 was chosen. The results from program 14 are compared with those of the N.A.G. routine for ten eigenvalues in Table 5.6.2. The time taken by program 14 was 1610 seconds and the time taken by FO2AJA was 63 seconds.

EIGENVALUES FROM PROGRAM 14		EIGENVALUES FROM FO2AJA	
REAL	IMAGINARY	REAL	IMAGINARY
-9.010258685	0.000000000	-9.010258684	0.000000000
-0.6435526709	0.000000000	-0.6435526712	0.000000000
0.5473984273	-0.06227686443	0.5473984272	-0.06227686378
0.5473984273	0.06227686442	0.5473984272	0.06227686378
9.307845570	0.000000000	9.307845573	0.000000000
10.38549187	0.000000000	10.38549186	0.000000000
11.49296839	0.000000000	11.49296838	0.000000000
15.79316104	0.000000000	15.79316104	0.000000000
16.82574875	0.000000000	16.82574875	0.000000000
19.67294860	0.000000000	19.67294860	0.000000000

TABLE 5.6.2

CHAPTER 6

THE DETERMINATION OF STURM SEQUENCES FOR SPARSE BANDED MATRICES

6.1 INTRODUCTION

Using methods similar to those of Chapter 5 it is possible to evaluate the Sturm sequences for sparse matrices with a more complex structure than those already described, and so determine a recursive formula for the sequence. These formulae can then be used in a bisection process or Muller's method to determine the eigenvalues of the matrix.

No algorithm is given for finding the eigenvectors of each matrix except for the periodic quindiagonal matrix as it is believed that the method described by Martin and Wilkinson (1967), and Wilkinson (1972) are the best available. The reason for this is that as the matrix becomes less sparse, the complexity of programming to utilise the sparseness is prohibitive except in the periodic quindiagonal case. The state is reached where, for large matrices, the space saved is small, and this is taken up by the extra program steps required to carry out the algorithm. Also the speed in calculating the vector is not critical, as, compared to the time taken to obtain an eigenvalue it is very small.

6.2 STURM SEQUENCES FOR THE QUINDIAGONAL AND PERIODIC QUINDIAGONAL MATRICES

The Sturm sequence for a quindiagonal matrix has already been found by Evans (1975), in the form of a sequence of leading principal minors of the matrix, obtained by Laplace expansion. The quindiagonal matrix is,

$$A = \begin{bmatrix} c_1, & b_2, & a_3, & & & & & & \\ b_2, & c_2, & b_3, & a_4, & & & & & \\ a_3, & b_3, & c_3, & \ddots & \ddots & & 0 & & \\ & a_4, & b_4, & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & a_N & \\ & & & \ddots & \ddots & \ddots & \ddots & b_N & \\ & & 0 & & \ddots & \ddots & \ddots & c_N & \end{bmatrix} \qquad (6.2.1)$$

and the sequence obtained for the matrix $(A-\lambda I)$ is,

$$
\left.
\begin{aligned}
P_0 &= 1 \quad , \\
P_1 &= (c_1-\lambda)P_0 \quad , \\
P_2 &= (c_2-\lambda)P_1 - b_2^2 P_0 \quad , \\
P_i &= (c_i-\lambda)P_{i-1} - b_i^2 P_{i-2} - a_i^2((c_{i-1}-\lambda)P_{i-3} - a_{i-1}^2 P_{i-4}) \\
&\quad + 2a_i b_i (b_{i-1}P_{i-3} - b_{i-2}q_{i-1}P_{i-4} + b_{i-3}a_{i-1}a_{i-2}P_{i-5} - \ldots .) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i=3,4,\ldots,N-1, \\
P_N &= (c_N-\lambda)P_{N-1} - b_N^2 P_{N-2} - a_N^2((c_{N-1}-\lambda)P_{N-3} - a_{N-1}^2 P_{N-4}) \\
&\quad + 2 \sum_{j=1}^{N-2} (-1)^{j+1} b_N b_{N-j} \left[\prod_{r=N-j+1}^{N} a_r \right] P_{N-J+2}
\end{aligned}
\right\} \quad (6.2.2)
$$

The sequence is also given as a sequence of the ratios of the minors to avoid underflow and overflow on a computer.

$$
\left.
\begin{aligned}
p_0 &= 1 \quad , \\
p_1 &= (c_1-\lambda) \quad , \\
p_2 &= (c_2-\lambda) - b_2^2/p_1 \quad , \\
p_3 &= (c_3-\lambda) - b_3^2/p_2 - a_3^2(c_2-\lambda)/(p_2p_1) + 2a_3b_2b_3/(p_2p_1), \\
&\vdots \\
p_N &= (c_N-\lambda) - b_N^2/p_{N-1} - a_N^2((c_{N-1}-\lambda)/(p_{N-1}p_{N-2}) - a_{N-1}^2/(p_{N-1}p_{N-2}p_{N-3})) \\
&\quad + 2 \sum_{j=1}^{N-2} (-1)^{j+1} b_N b_{N-j} \left[a_N/(p_{N-j-1}p_{N-j}) \prod_{r=N-j+1}^{N-1} (a_r/p_r) \right]
\end{aligned}
\right\} \quad (6.2.3)
$$

The quindiagonal matrix, and the periodic tridiagonal matrix can be seen to be special cases of the banded quindiagonal matrix given in Chapter 5 with P=3 or N. As has been shown the sequence (6.2.3) is equivalent to a sequence obtained from the matrix by elimination with no interchanges. This fact can be proved directly by a similar method to that employed in Chapter 3, but involves a lot of difficult algebra and serves no useful purpose, so is not included.

The matrix (A-λI) after the elimination process has the form

$$\begin{bmatrix} q_1 & r_2 & s_3 & & & & \\ & q_2 & r_3 & s_4 & & 0 & \\ & & q_3 & r_4 & s_5 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & & & s_N \\ & & & & & & r_N \\ & & & & & & q_N \end{bmatrix} , \qquad (6.2.5)$$

where,

$$\left.\begin{aligned} s_i &= a_i \ , \\ r_2 &= b_2 \ , \\ r_i &= b_i - s_i r_{i-1}/q_{i-2} \ , \quad i=3,4,\ldots,N \ , \\ \\ q_1 &= c_1 - \lambda \ , \\ q_2 &= c_2 - \lambda - r_2^2/q_1 \ , \\ q_i &= c_i - \lambda - r_i^2/q_{i-1} - s_i^2/q_{i-2}, \quad i=3,4,\ldots,N \ , \end{aligned}\right\} \qquad (6.2.6)$$

The sequence q_i, i=1,2,...,N can now be used to isolate eigenvalues in a bisection process, in place of the P_i, i=1,2,...,N.

Evans (1975) has also given the Sturm sequence for an unsymmetric matrix and the differentiated sequence for use in conjunction with a Newton method. These can be obtained from an unsymmetric matrix direct, or by setting P=3 in section 5.6, and using Muller's method. The matrix most similar in derivation to the quindiagonal matrix is the periodic quindiagonal matrix which has six additional elements in the top right and bottom left hand corners.

The matrix has the general form

$$C = \begin{bmatrix} c_1 & b_2 & e_3 & & & & d_1 & d_2 \\ b_2 & c_2 & b_3 & & & & & d_3 \\ e_3 & b_3 & c_3 & & & & & \\ & & & \ddots & & 0 & & \\ & & & & \ddots & & & \\ & & & & & & & e_N \\ d_1 & & & 0 & & & & b_N \\ d_2 & d_3 & & & & e_N & b_N & c_N \end{bmatrix} . \qquad (6.2.8)$$

Gaussian elimination can now be performed without interchanges on $(C-\lambda I)$ and the resulting matrix is,

$$\begin{bmatrix} U_1 & V_2 & e_3 & & & & X_1 & Y_1 \\ & U_2 & V_3 & e_4 & & 0 & X_2 & Y_2 \\ & & U_3 & & \ddots & & \vdots & \vdots \\ & & & & & e_{N-2} & X_{N-4} & \\ & & & & & & X_{N-3} & Y_{N-3} \\ & & & & & & & Y_{N-2} \\ & & & 0 & & & & V_N \\ & & & & & & & U_N \end{bmatrix} \qquad (6.2.9)$$

where,

$$\left.\begin{aligned} U_1 &= c_1-\lambda \quad , \\ V_2 &= b_2 \quad , \\ U_2 &= c_2-\lambda - V_2^2/U_1 \quad , \\ V_i &= b_i-e_iV_{i-1}/U_{i-1} \quad , \quad i=3,4,\ldots,N-2, \\ U_i &= c_i-\lambda-V_i^2/U_{i-1}-e_i^2/U_{i-2} \quad , \quad i=3,4,\ldots,N-2, \\ X_1 &= d_1,\ X_2 = -V_2X_1/U_1 \quad , \\ Y_1 &= d_2,\ Y_2 = d_3-V_2Y_1/U_1 \quad , \\ X_i &= -X_{i-1}V_i/U_{i-1}-X_{i-2}e_i/U_{i-2} \quad , \quad i=3,4,\ldots,N-3, \\ Y_i &= -Y_{i-1}V_i/U_{i-1}-Y_{i-2}e_i/U_{i-2} \quad , \quad i=3,4,\ldots,N-2, \\ X_{N-3} &= e_{N-3}-X_{N-3} \quad , \end{aligned}\right\} \qquad (6.2.10)$$

$$\left.\begin{aligned} V_{N-1} &= b_{N-1} - X_{N-3}V_{N-2}/U_{N-2} \quad , \\ U_{N-1} &= c_{N-1} - \lambda - \sum_{i=1}^{N-3} X_i^2/U_i - V_{N-1}^2/U_{N-2} \quad , \\ V_N &= b_N - \sum_{i=1}^{N-2} V_i^2/U_i \quad , \\ U_N &= c_N - \lambda - \sum_{i=1}^{N-2} Y_i^2/U_i - V_N^2/U_{N-1} \quad , \end{aligned}\right\} \qquad (6.2.10)$$

This recursive sequence is simple and easy to calculate and the U_i, $i=1,2,\ldots,N$ can be used to isolate the eigenvalues of the matrix C in a bisection process.

There are two different strategies that can be used here to optimise one of two factors when calculating the eigenvalues on a computer. The bisection algorithm as given by Barth et al (1967) and described in Appendix 1 can be used. Here the whole sequence is calculated for every bisection. The sequence given by (6.2.10) can then be calculated in the order,

$$\left.\begin{aligned} &U_1 \ , X_1 \ , Y_1 \ , && (U_N,V_N,V_{N-1},U_{N-1}) \ , \\ &U_2 \ , V_2 \ , X_2 \ , Y_2 \ , && (U_N,V_N,V_{N-1},U_{N-1}) \ , \\ &U_3 \ , V_3 \ , X_3 \ , Y_3 \ , && (U_N,V_N,V_{N-1},U_{N-1}) \ , \\ &\vdots && \vdots \\ &U_{N-3}, V_{N-3}, X_{N-3}, Y_{N-3}, && (U_N,V_N,V_{N-1},U_{N-1}) \ , \\ &U_{N-2}, V_{N-2}, Y_{N-2}, \\ &V_{N-1}, U_{N-1}, V_N \ , U_N \ . \end{aligned}\right\} \qquad (6.2.11)$$

and therefore only eight extra storage locations are required to calculate the sequence from the given matrix. This procedure is slower than the procedure described in Chapter 5 where at each stage the sequence is calculated only as far as is needed for that particular eigenvalue. In this case the elements of the sequence (6.2.10) are calculated in the

following order,

$$\left.\begin{array}{l} U_1 \ , V_2 \ , \\ U_2 \ , V_3 \ , \\ \vdots \quad \vdots \\ U_{N-3}, V_{N-2}, \\ U_{N-2}, \\ X_1 \ , X_2 \ , X_3, \ldots\ldots\ldots, X_{N-3}, V_{N-1}, U_{N-1}, \\ Y_1 \ , Y_2 \ , Y_3, \ldots\ldots\ldots, Y_{N-2}, V_N, U_N, \end{array}\right\} \qquad (6.2.12)$$

and therefore a further 2N storage locations in the computer are required to calculate this sequence from the given matrix.

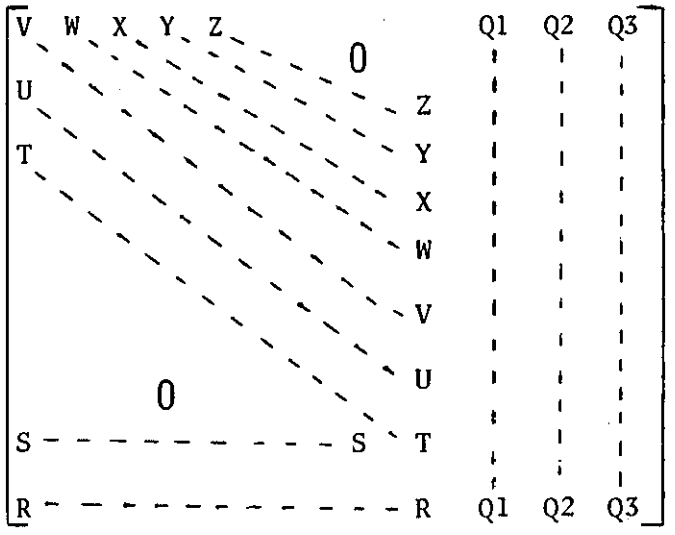
So that if, for a large matrix, the eigenvalues are to be found the algorithm can be chosen to minimise storage requirements or time taken depending on the machine loading. A similar procedure can be adopted with the periodic tridiagonal matrix described in Chapter 3. The Meteorological Office at present are using a similar procedure to that just described. Here they have large numbers of periodic tridiagonal matrices to solve whilst predicting weather trends. They have two algorithms available, the one already described by Evans, and one given by Golub(1973), and choose which one to use depending on whether they wish to minimise space used or time taken.

The two procedures to determine the eigenvalues of the periodic quindiagonal are given in Appendix 1 in program 15 and 16. The test results are included in 6.4.

6.3 EIGENVECTORS OF A SYMMETRIC PERIODIC QUINDIAGONAL MATRIX

The eigenvectors of this matrix (6.2.8) can be found using inverse iteration in a manner that is both efficient and saves storage. The method is the same as that used for a banded quindiagonal matrix, and a periodic tridaigonal matrix, and is described briefly here.

The matrix, which is stored as three vectors, has the appropriate eigenvalue subtracted from the diagonal elements and is stored in 12 vectors to represent the matrix ready for elimination.



(6.3.1)

where, from (6.2.8)

$$\left.\begin{aligned} V &= c-\lambda \quad , \\ U &= W = b \quad , \\ T &= X = e \quad , \\ Y &= Z = Q1 = 0 \quad , \\ S_1 &= Q2_1 = d_1 \quad , \\ R_1 &= Q3_1 = d_2 \quad , \\ R_2 &= Q3_2 = d_3 \quad , \\ R &= S = Q2 = Q3 = 0 \quad , \end{aligned}\right\} \qquad (6.3.2)$$

Now the first stage of inverse iteration can be performed, that is, the elimination to upper triangular form.

At the i^{th} stage (say) there are a possible four interchanges of V_i with U_{i+1}, T_{i+2}, S_i, or R_i to choose from or no interchange at all. This interchange information is stored in the interchange vector in the i^{th} position with either the row number the i^{th} row is interchanged with or a zero (no interchange). So that if T_{i+2} was the element with largest modulus then the following would be interchanged

$$\left.\begin{aligned} V_i &\leftrightarrow T_{i+2} \ , \\ W_{i+1} &\leftrightarrow U_{i+2} \ , \\ X_{i+2} &\leftrightarrow V_{i+2} \ , \\ Y_{i+3} &\leftrightarrow W_{i+3} \ , \\ Z_{i+4} &\leftrightarrow X_{i+3} \ , \\ Q1_i &\leftrightarrow Q1_{i+2} \ , \\ Q2_i &\leftrightarrow Q2_{i+2} \ , \\ Q3_i &\leftrightarrow Q3_{i+2} \ , \end{aligned}\right\} \qquad (6.3.3)$$

and the interchange vector would be set,

$$IC_i = i+2 \ .$$

Now the elements U_{i+1}, T_{i+2}, S_i, and R_i are to be eliminated. As an example the element T_{i+2} will be dealt with in detail. First the elimination factor is calculated (T_{i+2}/V_i) and stored in T_{i+2}. Then the i^{th} row times the elimination factor is subtracted from the remainder of the $i+2^{th}$ row,

$$\left.\begin{aligned} U_{i+2} &= U_{i+2}-W_{i+1}T_{i+2} \ , \\ V_{i+2} &= V_{i+2}-X_{i+2}T_{i+2} \ , \\ W_{i+3} &= W_{i+3}-Y_{i+3}T_{i+2} \ , \\ X_{i+3} &= X_{i+3}-Z_{i+4}T_{i+2} \ , \\ Q1_{i+2} &= Q1_{i+2}-Q1_iT_{i+2} \ , \\ Q2_{i+2} &= Q2_{i+2}-Q2_iT_{i+2} \ , \\ Q3_{i+2} &= Q3_{i+2}-Q3_iT_{i+2} \ , \end{aligned}\right\} \qquad (6.3.4)$$

This process is continued until the last seven rows where care has to be taken over which elements are still present. Then, the eigenvector is set to all 1's and the back substitution takes place. Again care has to be taken over the initial elements, but if the eigenvector is $\underline{Q}$ then at the i^{th} step the back substitution has the form,

$$\begin{aligned} Q_i = (Q_i &-Q3_iQ_N-Q2_iQ_{N-1}-Q1_iQ_{N-2}-W_{i+1}Q_{i+1} \\ &-X_{i+2}Q_{i+2}-Y_{i+3}Q_{i+3}-Z_{i+4}Q_{i+4})/V_i \ . \end{aligned} \qquad (6.3.5)$$

As indicated by Wilkinson (1965), two steps of inverse iteration will usually gain full accuracy provided there is an accurate approximation to an eigenvalue. So the second step is started, and this time, only the stored interchanges and elimination factors are used. So that at the i^{th} stage IC_i is equal to i+2 so two elements are interchanged,

$$Q_i \leftrightarrow Q_{i+2} \quad , \tag{6.3.6}$$

and when the elimination is carried out for the $i+2^{th}$ row only,

$$Q_{i+2} = Q_{i+2} - Q_i T_{i+2} \quad . \tag{6.3.7}$$

Again care must be taken with the last stages, but when this has been completed the back substitution is performed the same as in (6.3.5). At the end of this $\underline{Q}$ contains the desired eigenvector.

The program to perform this algorithm is given in Appendix 1 in program 17. The results for this and the preceding section are given in section 6.4.

It should be noted that 13N extra storage locations are required for the algorithm, and that due to its complexity no storage savings are made unless the matrix is large (>30×30). However it is computed in an efficient fashion with no double indexing and runs efficiently compared to any other method regardless of the size of the matrix.

6.4 RESULTS

As no useful periodic quindiagonal matrices with known eigenvalues could be found results were compared with those obtained by a N.A.G. routine. These are given later, and are found to agree to 10 significant figures.

First for a number of matrices of different sizes the three variations of the algorithm (normal, space saving, time saving) were run, on the ICL 1904S, and the times taken compared, the results are given in the following

table (6.4.1).

	TIME TAKEN IN SECONDS TO OBTAIN ALL EIGENVALUES		
MATRIX SIZE	NORMAL ALGORITHM	SPACE SAVING ALGORITHM PROGRAM 15	TIME SAVING ALGORITHM PROGRAM 16
30 × 30	30	29	27
50 × 50	92	84	78
80 × 80	237	218	174
90 × 90	308	283	234
100 × 100	381	351	296
110 × 110	465	427	342

TABLE 6.4.1.

The normal algorithm is the bisection algorithm of Barth et al, (1967) coupled with the inefficient storage method of (6.2.12) for calculating the Sturm sequence of the matrix. The space saving algorithm uses the same bisection algorithm as before, but avoids using arrays in calculating the Sturm sequence of the matrix. Due to the fact that no array accesses are made the space saving algorithm is slightly more efficient than the normal algorithm. As can be seen from the table (6.4.1) the time efficient algorithm makes savings in time of up to 30% on the normal algorithm, and this figure increases with larger matrices. Even for a very large matrix of size 1000×1000 (say), the total extra storage for the time efficient algorithm

is 2000 computer words. This is a small enough figure compared to the core size of most computers to make this the preferred algorithm in most applications.

A test matrix of order 50 is used to illustrate the results and the resultant eigenvalues, and eigenvectors are compared with those obtained using a N.A.G. library routine. The library routine used was F02ABA which uses a Householders reduction and QL algorithm.

The test matrix has a simple form consisting of all 1's for the diagonal elements, all 2's for the sub-diagonal elements, all 1's for the sub-sub-diagonal elements, and the three corner elements all 0.5. The matrix is symmetric, and the 50 eigenvalues are all contained in the range $(-5 \leq \lambda \leq 7)$. The ten smallest eigenvalues are given in table (6.4.2) and the eigenvector corresponding to the tenth in table (6.4.2) in table (6.4.3).

EIGENVALUES-PROGRAM 16	EIGENVALUES-F02ABA
-1,989932556	-1,989932555
-1.988566522	-1.988566522
-1.972569410	-1.972569410
-1.955757440	-1.955757440
-1.913955008	-1.913955008
-1.900012702	-1.900012702
-1.856065127	-1.856065127
-1.825883875	-1.825883875
-1.778871004	-1.778871004
-1.737621856	-1.737621856

TABLE 6.4.2

EIGENVECTOR ELEMENTS x_1-x_{25} PROGRAM 16	EIGENVALUE ELEMENTS x'_1-x'_{25} F02ABA	EIGENVECTOR ELEMENTS x_{26}-x_{50} PROGRAM 16	EIGENVECTOR ELEMENTS x'_{26}-x'_{50} F02ABA
$9.123624343*10^{-2}$	$9.123524342*10^{-2}$	$-2.417438794*10^{-1}$	$-2.417438795*10^{-1}$
$-3.169860411*10^{-2}$	$-3.169860406*10^{-2}$	$3.052577011*10^{-1}$	$3.052577006*10^{-1}$
$-1.566043062*10^{-1}$	$-1.566043063*10^{-1}$	$1.477897989*10^{-1}$	$1.477897991*10^{-1}$
$2.631330387*10^{-1}$	$2.631330388*10^{-1}$	$-1.374037339*10^{-1}$	$-1.374037340*10^{-1}$
$-1.253817412*10^{-1}$	$-1.253817412*10^{-1}$	$5.090722328*10^{-2}$	$5.090722330*10^{-2}$
$2.229621417*10^{-1}$	$2.229621417*10^{-1}$	$9.117364089*10^{-3}$	$9.117364071*10^{-3}$
$-1.169450844*10^{-1}$	$-1.169450844*10^{-1}$	$-5.057730172*10^{-1}$	$-5.057730128*10^{-1}$
$-1.738004344*10^{-3}$	$-1.738004371*10^{-3}$	$-1.868700158*10^{-1}$	$-1.868700157*10^{-1}$
$2.391203200*10^{-3}$	$2.391203237*10^{-3}$	$2.569992520*10^{-1}$	$2.569992520*10^{-1}$
$1.090361934*10^{-2}$	$1.090361928*10^{-2}$	$-8.344772451*10^{-2}$	$-8.347724515*10^{-2}$
$9.206764423*10^{-2}$	$9.206764425*10^{-2}$	$-1.578145173*10^{-1}$	$-1.578145174*10^{-1}$
$-2.170296771*10^{-1}$	$-2.170296771*10^{-1}$	$2.170296771*10^{-1}$	$2.170296773*10^{-1}$
$1.578145173*10^{-2}$	$1.578145172*10^{-2}$	$-9.206764421*10^{-2}$	$-9.206764431*10^{-2}$
$8.347724515*10^{-1}$	$8.347724523*10^{-1}$	$-1.090361936*10^{-3}$	$-1.090361943*10^{-3}$
$-2.569992520*10^{-1}$	$-2.569992520*10^{-1}$	$-2.391203193*10^{-3}$	$-2.391203031*10^{-3}$
$1.868700158*10^{-1}$	$1.868700157*10^{-1}$	$1.738004355*10^{-3}$	$1.738004237*10^{-3}$
$5.057730177*10^{-3}$	$5.057730212*10^{-3}$	$1.169450844*10^{-1}$	$1.169450844*10^{-1}$
$-9.117364090*10^{-2}$	$-9.117364091*10^{-2}$	$-2.229621417*10^{-1}$	$-2.229621416*10^{-1}$
$5.176034959*10^{-2}$	$5.176034963*10^{-2}$	$1.246880589*10^{-1}$	$1.246880588*10^{-1}$
$-5.090722328*10^{-1}$	$-5.090722331*10^{-1}$	$1.253817412*10^{-1}$	$1.253817413*10^{-1}$
$1.374037339*10^{-1}$	$1.374037338*10^{-1}$	$-2.631330387*10^{-1}$	$-2.631330388*10^{-1}$
$-1.477897989*10^{-1}$	$-1.477897988*10^{-1}$	$1.566043062*10^{-1}$	$1.566043062*10^{-1}$
$-3.052577012*10^{-2}$	$-3.052577021*10^{-2}$	$3.169860414*10^{-2}$	$3.169860424*10^{-2}$
$2.417438794*10^{-1}$	$2.417438795*10^{-1}$	$-9.123624345*10^{-2}$	$-9.123524351*10^{-2}$

TABLE 6.4.3

The results for the two programs are in agreement to at least 10 significant figures for most values. This test matrix is also centro-symmetric, which means the eigenvector (in this case) is anti-centro-symmetric Therefore the sum of two elements x_i, x_j (where i+j-1=50) should be zero. In this respect the results from program 16 were more consistent than those from the N.A.G. routine.

6.5 STURM SEQUENCES FOR FURTHER BANDED SYSTEMS

The next matrix for which the eigenvalues are determined is not one found commonly in practice, but the sequence is given, as it was used to provide a convenient "stepping stone" to determine the sequence of a more useful and more *dense* matrix in a later section.

The matrix in question has the form:

$$C = \begin{bmatrix} c_1 & b_2 & d_3 & & & e_P & & 0 \\ b_2 & c_2 & b_3 & & & & \ddots & \\ d_3 & b_3 & & & & 0 & & e_N \\ & & & & & & & \\ & & & & & & & d_N \\ e_P & & 0 & & & & & b_N \\ & \ddots & & & & & & \\ 0 & & e_N & & d_N & b_N & & c_N \end{bmatrix} \qquad (6.5.1)$$

Now in order to determine a Sturm sequence for this matrix the same procedure as used in previous chapters is carried out. Gaussian elimination without pivoting is performed, and careful note taken of how each element is produced then, with judicious relabelling of the elements in the remaining upper triangular matrix, a matrix of the following form is left,

$$\begin{bmatrix} R_{1,1} & R_{2,2} & R_{3,3} & & & R_{P,P} & & \\ & R_{2,1} & R_{3,2} & & 0 & R_{P,P-1} & R_{P+1,P} & 0 \\ & & R_{3,1} & \ddots & & \vdots & \ddots & \ddots \\ & & & \ddots & & & & R_{N,P} \\ & & & & & R_{P,2} & & R_{N,P-1} \\ & & & & & R_{P,1} & & \\ & & 0 & & & & & R_{N,3} \\ & & & & & & & R_{N,2} \\ & & & & & & & R_{N,1} \end{bmatrix} \qquad (6.5.2)$$

Where the $R_{i,j}$ elements are described by the following recursive formula:

$$\left.\begin{aligned} R_{1,1} &= c_1-\lambda \quad , \quad R_{2,2} = b_2 \\ R_{2,1} &= c_2-\lambda-R_{2,2}R_{2,2}/R_{1,1} \\ R_{i,3} &= d_i \qquad\qquad i=3,4,\ldots,P-1, \\ R_{i,P} &= e_i \qquad\qquad i=P,P+1,\ldots,N, \end{aligned}\right\} \qquad (6.5.3)$$

$$\left.\begin{array}{l}
R_{i,2} = b_i - R_{i-1,2}R_{i,3}/R_{i-2,1} \qquad\qquad ,i=3,4,\ldots,P-1, \\
R_{i,1} = c_i - \lambda - R_{i,2}R_{i,2}/R_{i-1,1} - R_{i,3}/R_{i-2,1} \qquad ,i=3,4,\ldots,P-1, \\
R_{i,k} = \varepsilon_k - \sum_{j=0}^{i-k+1} R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+J+1,1} \\
\qquad\qquad i=P,P+1,\ldots,N, \quad k=P-1,P-2\ldots 1, \\
\text{where} \\
\varepsilon_{i,\ell} = \begin{cases} e_i \text{ if } = P \\ d_i \text{ if } = 3 \\ b_i \text{ if } = 2 \\ c_i-\lambda \text{ if } \ell = 1 \\ 0 \text{ otherwise} \end{cases} \qquad i=P,P+1,\ldots,N ,
\end{array}\right\} \quad (6.5.3)$$

The sequence has been programmed in ALGOL 60 and the program is given in Appendix 1 in program 18. No results are given as the matrix is of no great practical importance, but the exercise was carried out to gain manipulative skills to cope with more difficult banded systems to come.

The next matrix to be considered is one of more practical importance, and has the form,

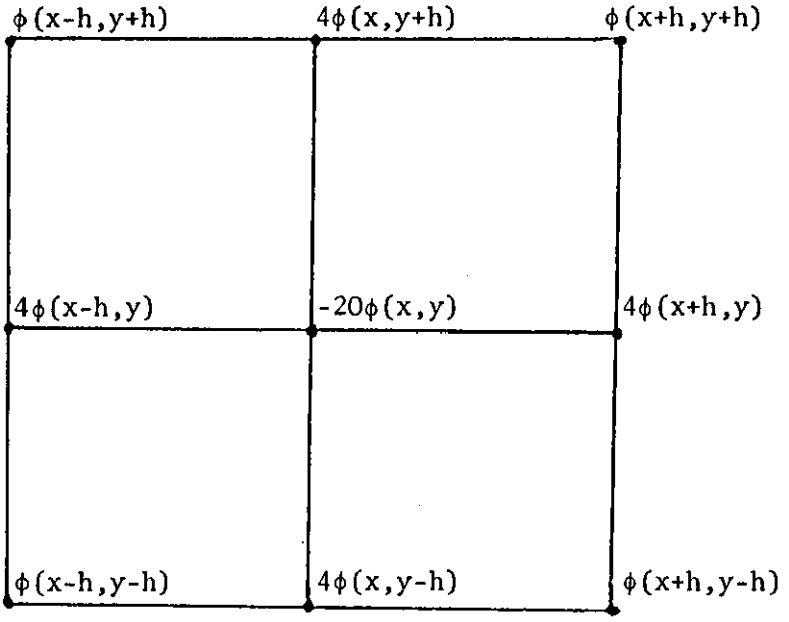
$$C = \begin{bmatrix}
c_1 & b_2 & & & d_{P,P} & d_{P+1,P+1} & \cdots & d_{M,M} & & 0 \\
b_2 & c_2 & \ddots & & & d_{P+1,P} & \cdots & d_{M,M-1} & \ddots & \\
 & \ddots & \ddots & \ddots & & & \ddots & & \ddots & d_{N,M} \\
 & & \ddots & \ddots & \ddots & & 0 & & \ddots & \vdots \\
d_{P,P} & & & \ddots & \ddots & \ddots & & & & d_{N,P} \\
d_{P+1,P+1} & \ddots & & 0 & & \ddots & \ddots & & & \\
\vdots & & \ddots & & & & \ddots & \ddots & & b_N \\
d_{M,M} & & & \ddots & & & & \ddots & b_N & c_N \\
0 & & \ddots & d_{N,M} & \cdots & d_{N,P} & & & &
\end{bmatrix}$$

(6.5.4)

This is a block banded matrix of semi bandwidth M with a block of width M-P+1.

Typically, a matrix of the form (6.5.4) can arise when a finite difference approximation to the second order partial differential equation of section 5.2 is used. If, instead of the five point formula, the more accurate nine point formula is used, a matrix similar to (6.5.4) arises where M-P+1 is equal to three.

The computational molecule applied to the grid of figure (5.2.2) to give the nine point formula is given in figure (6.5.1)



COMPUTATIONAL MOLECULE FOR 9 POINT FORMULA

FIGURE 6.5.1

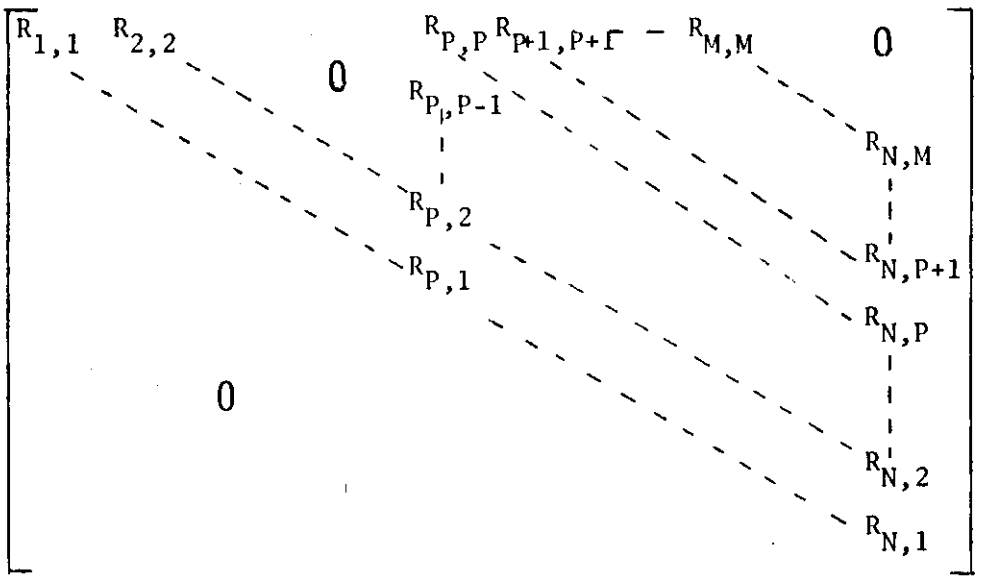
In the notation of section 5.2 the nine point formula equivalent to (5.2.1f) can be given as

$$-(\phi(x+h,y-h)+\phi(x-h,y-h)+\phi(x+h,y+h)+\phi(x-h,y+h)-20(x,y)$$
$$+4\phi(x-h,y)+4\phi(x+h,y)+4\phi(x,y+h)+4\phi(x,y-h)) = \lambda\phi(x,y) \ . \qquad (6.5.4a)$$

The application of this formula at all the grid points of the system produces the N (say) simultaneous linear equations which can be written in matrix form as,

$$C\underline{x} = \lambda\underline{x}$$

Again the Sturm sequence for the matrix C is found by performing Gaussian elimination on the matrix (C-λI) without interchanges. Then by noting how each element was formed in the remaining upper triangular matrix and by relabelling the elements a matrix is left of the form,



(6.5.5)

Where the R elements are obtained by using the following recursive formula,

$$\left.\begin{aligned}
R_{1,1} &= c_1-\lambda && ,\\
R_{i,1} &= c_i-\lambda-b_i^2/R_{i-1,1} && ,i=2,3,\ldots,P-1,\\
R_{i,i} &= d_{i,i} && ,i=P,P+1,\ldots M,\\
R_{i,M} &= d_{i,M} && ,i=M,m+1\ldots,N,\\
R_{i,j} &= 0 && ,i<j\\
R_{i,k} &= \ell_{i,k} - \sum_{j=0}^{i-k+1} R_{i,i-j}R_{i-k-1,i-k+1-j}/R_{j+1} && \\
& && i=P,P+1\ldots.M,\ k=i-1,i-2,\ldots 1,\\
R_{i,k} &= \ell_{i,k} - \sum_{j=0}^{M-k+1} R_{i,M-j}R_{i-k+1,M+1-k-j}/R_{i-M+J+1,1}, && \\
& && i=M,M+1,\ldots N,\ k=M-1,M-2,\ldots 1,\\
\ell_{i,k} &= \begin{cases} c_i, & \text{if } k=1\\ b_i, & \text{if } k=2\\ d_{i,k}, & \text{if } k\geq P\\ 0 & \text{otherwise}\end{cases} && i=P,P+1,\ldots N,
\end{aligned}\right\} \quad (6.5.6)$$

The $R_{i,1}$ i=1,N can now be used in a bisection process to isolate the eigenvalues of matrix C. The algorithm to perform this is given in program 19 in Appendix 1.

There is no associated algorithm given for the eigenvectors of this matrix. The reason for this is that the complexity of programming the method to take advantage of the sparsity of the matrix, far outweighs any gains that may be made. Therefore the method described by Martin et al. (1972) is recommended for use in this case. This method is a more general Inverse Iteration procedure that loses little when used on matrices of the type (6.5.4).

Once the recursive sequence (6.5.6) had been found, it was an easy step to find the sequence for another type of matrix closely related in structure to (6.5.1) and (6.5.4). This matrix has the form,

$$C=\begin{bmatrix}
c_1 & b_2 & e_3 & & & d_{P,P} & d_{P+1,P+1} & \cdots & d_{M,M} & & \\
b_2 & c_2 & b_3 & e_4 & & & d_{P+1,P} & & & 0 & \\
e_3 & b_3 & & & & & & & & & d_{M,N} \\
 & e_4 & & & & 0 & & & & & d_{P+1,N} \\
 & & & & & & & & & & d_{P,N} \\
d_{P,P} & & & & & & & & & & \\
 & & & & & & & & & & \\
d_{M,M} & & & 0 & & & & & & & e_N \\
 & & & & & & & & & & b_N \\
 & 0 & & & & & & & & & c_N \\
 & & d_{M,N} & \cdots & d_{P,N} & & & e_N & b_N & &
\end{bmatrix} \qquad (6.5.7)$$

A typical problem from an engineering application that gives rise to a matrix of the form (6.5.7) occurs when determining the natural frequencies of modes of a clamped, square plate. The natural frequencies and modes of vibration of the clamped elastic plate are obtained from the eigenvalues λ

and eigenvectors $\omega(x,y)$ of,

$$\frac{\partial^4\omega}{\partial x^4} + \frac{2\partial^4\omega}{\partial x^2\partial y^2} + \frac{\partial^4\omega}{\partial y^4} = \lambda\omega \quad , \tag{6.5.8}$$

where x,y are in R, and

$$\omega = \frac{\partial\omega}{\partial x} = 0 \quad , \tag{6.5.9}$$

for x,y on B.

R is the region of the square plate and (6.5.9) defines the boundary conditions on the boundary (B) for clamping on B.

By setting up a grid across the plate and applying the thirteen point difference formula approximation to equation (6.5.7) at all the grid points, a set of N (say) linear homogeneous equations are formed. The computational molecule for the thirteen point formula is given in figure (6.5.2).

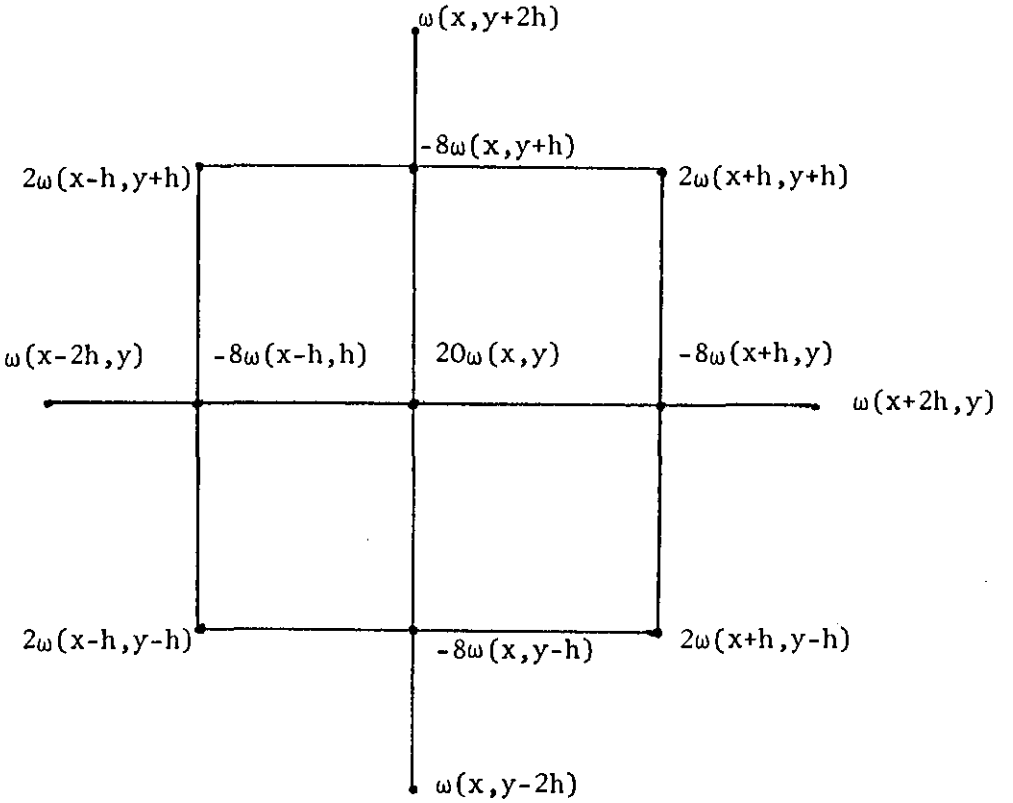


FIGURE 6.5.2

The equations obtained from the approximation to (6.5.8) can be arranged so that they can be expressed in matrix form as,

$$C\underline{\omega} = \lambda\underline{\omega} \quad ,$$

where C is defined by (6.5.6). The problem is then that of finding the eigenvalues and eigenvectors of the matrix C.

Again the practical procedure is to reduce the matrix (C-λI) to upper triangular form by Gaussian elimination without pivoting and a careful note taken of how the elements in the matrix were formed. The resulting matrix has the form,

$$\begin{bmatrix} R_{1,1} & & & & R_{P,P} & R_{P+1,P+1} & \cdots & R_{M,M} & & 0 \\ & R_{2,2} & & 0 & R_{P,P-1} & & & & & \\ & & R_{3,3} & & & & & & & R_{M,N} \\ & & & & R_{P,3} & & & & & \\ & & & & R_{P,2} & & & & & R_{P+1,N} \\ & & & & R_{P,1} & & & & & \\ & & & & & & & & & R_{P,N} \\ & & & 0 & & & & & & \\ & & & & & & & & & R_{N,3} \\ & & & & & & & & & R_{N,2} \\ & & & & & & & & & R_{N,1} \end{bmatrix} \tag{6.5.10}$$

where the R elements are defined by the following recursive relationships,

$$\left.\begin{aligned} R_{1,1} &= c_1-\lambda \\ R_{2,2} &= b_2 \\ R_{2,1} &= c_2-R_{2,2}R_{2,2}/R_{1,1} \\ R_{i,3} &= e_i && , \ i=3,4,\ldots,P-1, \\ R_{i,i} &= d_{i,i} && , \ i=P,P+1,\ldots M, \\ R_{i,M} &= d_{i,M} && , \ i=M,M+1,\ldots,N, \\ R_{i,2} &= b_i-R_{i-1,2}R_{i,3}/R_{i-2,1} \\ R_{i,1} &= c_i-\lambda-R_{i,2}R_{i,2}/R_{i-1,1}-R_{i,3}R_{i,3}/R_{i-2,1} \end{aligned}\right\} \ i=3,4,\ldots,P-1, \tag{6.5.11}$$

$$
\left.
\begin{aligned}
R_{i,k} &= \varepsilon_{i,k} - \sum_{j=0}^{i-k+1} R_{i,i-j} R_{i-k+1,I+1-K-J} / R_{j+1,1}, \\
&\qquad i=P,P+1,\ldots,M,\ k=i-1,i-2,\ldots 1, \\
R_{i,k} &= \varepsilon_{i,k} - \sum_{j=0}^{M-k+1} R_{i,M-j} R_{i-k+1,M+1-k-j} / R_{i-M+j+1,1}, \\
&\qquad i=M+1,M+2,\ldots,N,\ k=M-1,M-2,\ldots 1, \\
\varepsilon_{i,k} &= \begin{cases} c_i & \text{if } k=1, \\ b_i & \text{if } k=2, \\ e_i & \text{if } k=3, \\ d_{i,k} & \text{if } k \geq P, \\ 0 & \text{otherwise} \end{cases} \qquad i=P,P+1,\ldots,N,
\end{aligned}
\right\} \quad (6.5.11)
$$

Now the elements $R_{i,1}$, i=1,N can be used in a bisection process to isolate the eigenvalues of matrix C. The program to perform this algorithm is given in Appendix 1 in program 20.

6.6 THE DETERMINATION OF THE STURM SEQUENCE FOR A SYMMETRIC BANDED MATRIX AND THE ROUNDING ERROR ANALYSIS

The symmetric band matrix C with semi-bandwidth P is defined by,

$$
C = \begin{bmatrix}
c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & \cdots & c_{1,P} & & & & \\
c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & \cdots & c_{2,P} & c_{2,P+1} & & 0 & \\
c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & \cdots & c_{3,P} & c_{3,P+1} & c_{3,P+2} & & \\
 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & c_{N-P+1,N} \\
c_{P,1} & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & c_{N-1,N} \\
 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & c_{N,N} \\
 & & c_{N,N-P+1} & \cdots & \cdots & \cdots & \cdots & c_{N,N-1} & &
\end{bmatrix} \quad (6.6.
$$

Now by performing Gaussian elimination on the matrix (C-λI) without pivoting, and by relabelling elements as described in detail in Chapter 5 the matrix R is produced,

$$R = \begin{bmatrix} R_{1,1} & R_{2,2} & R_{3,3} & \cdots & R_{P,P} & & \\ R_{2,1} & R_{3,2} & R_{4,3} & \cdots & R_{P,P-1} & R_{P+1,P} & 0 \\ R_{3,1} & R_{4,2} & & \cdots & R_{P+1,P-1} & R_{P+1,P} & \\ & & & & & & R_{N,P} \\ & & & & & & R_{N,P-1} \\ & & 0 & & & & \vdots \\ & & & & & & R_{N,2} \\ & & & & & & R_{N,1} \end{bmatrix} \qquad (6.6.2)$$

Where the $R_{i,k}$, i=k...N, k=1,P are defined by the following recursive formula,

$$R_{i,k}(\lambda) = c_{i-k+1,i} - \sum_{j=0}^{i-k-1} R_{i,i-j} R_{i-k+1,i+1-k-j} / R_{j+1,1} \qquad i=1,\ldots P-1,\ k=1,\ldots i \qquad (6.6.3)$$

$$R_{i,k}(\lambda) = c_{i-k+1,i} - \sum_{j=0}^{P-k-1} R_{i,P-j} R_{i-k+1,P+1-k-j} / R_{i-P+j+1,1} \qquad i=P,\ldots,N,\ k=1,\ldots P \qquad (6.6.4)$$

The $R_{i,1}(\lambda)$, i=1,2,...N elements represent the ratios of the leading principal minors of the matrix (C-λI) and can now be used in a bisection process to eliminate the eigenvalues of C.

This method applied to this type of matrix is not competitive with other methods in finding eigenvalues quickly, and is not recommended as a useful method. However it is the most general form of a band matrix and is used to illustrate the rounding error analysis, which can then be

applied to the other types of band matrix.

In the analysis the notation of Wilkinson (1963) will be used, replacing 2^{-t} by ε. There are two cases to consider in the calculation of a $R_{i,k}$ and the zero in the symmetrically opposite position, for $i \geq P$ and $i<P$. Both cases are very similar and the case for $i \geq P$ (equation 6.6.4) is described.

The calculation of $R_{i,k}$ is done in P-k steps:

$$\left.\begin{aligned} R_{i,k}^{(0)} &= c_{i-k+1,i} \\ R_{i,k}^{(1)} &= R_{i,k}^{(0)} - R_{i,P}R_{i-k+1,P-k+1}/R_{i-P+1,1} + \varepsilon_{i,k}^{(1)} \\ R_{i,k}^{(2)} &= R_{i,k}^{(1)} - R_{i,P-1}R_{i-k+1,P-k}/R_{i-P+2,1} + \varepsilon_{i,k}^{(2)} \\ &\vdots \\ R_{i,k} &= R_{i,k}^{(P-k-1)} - R_{i,k+1}R_{i-k+1,2}/R_{i-k,1} + \varepsilon_{i,k}^{(P-k-1)} \end{aligned}\right\} \qquad (6.6.5)$$

Where all $R_{i,k}^{(j)}, R_{i,P-j}, R_{i-k+1,P+1-k-j}, R_{i-P+j+1,1}$ refer to computed values, and $\varepsilon_{i,k}^{(j)}$ is the difference between the accepted $R_{i,k}^{(j)}$ and the exact value which could be obtained using the computed values. Summing the equations (6.6.5) gives:

$$R_{i,k} = c_{i-k+1,i} - \sum_{j=0}^{P-k-1} R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,1} + \varepsilon_{i,k} \qquad (6.6.6)$$

where,

$$\varepsilon_{i,k} = \varepsilon_{i,k}^{(1)} + \varepsilon_{i,k}^{(2)} + \ldots\ldots + \varepsilon_{i,k}^{(P-k-1)} \qquad (6.6.7)$$

The production of a zero in the symmetrically opposite position to $R_{i,k}$ takes place in the same manner, except that when $R_{i,k}$ is obtained in that position it is set identically to zero. This is equivalent to performing the exact operation, and no error is involved. However this still produces errors in the modification of that row, due to the fact that $R_{i,P-j}/R_{i-P+j+1,1}$ will have a rounding error. This means that the error in producing a zero involves the extra term,

$$0 = R_{i,k} - R_{i,k}R_{i-k+1,1}/R_{i-k+1,1} + \overset{(P-k)}{\underset{i,k}{}}, \tag{6.6.8}$$

which is added to (6.6.5). Then the error in that position becomes,

$$\epsilon'_{i,k} = \epsilon^{(1)}_{i,k} + \epsilon^{(2)}_{i,k} + \ldots\ldots + \epsilon^{(P-k)}_{i,k} \tag{6.6.9}$$

Calculating $R^{(j+1)}_{i,k}$ (say) from (6.6.5) in floating point arithmetic produces,

$$R^{(j+1)}_{i,k} = f\ell(R^{(j)}_{i,k} - R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,1}) , \tag{6.6.10}$$

$$R^{(j+1)}_{i,k} = (R^{(j)}_{i,k} - R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,1}(1+\epsilon_2)(1+\epsilon_3))(1+\epsilon_1), \tag{6.6.11}$$

from which can be deduced:

$$R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,1} = (R^{(j)}_{i,k} - R^{(j+1)}_{i,k}(1+\epsilon_1)^{-1})(1+\epsilon_2)^{-1}(1+\epsilon_3)^{-1} \tag{6.6.12}$$

where $\epsilon_\ell < \epsilon$, $\ell=1,2,3$.

If

$$\left.\begin{aligned} \hat{R} &= \max_{\substack{i,k \\ j=0,1,\ldots,P-k-1}} |R^{(j)}_{i,k}| \\ \text{and} \\ \hat{\hat{R}} &= \max_{i,k} |R_{i,k}| \end{aligned}\right\} \tag{6.6.13}$$

and then

$$g = \max(\hat{R},\hat{\hat{R}}) \tag{6.6.14}$$

From (6.6.5) it can be seen that,

$$|\epsilon^{j+1}_{i,k}| = |R^{(j+1)}_{i,k} - R^{(j)}_{i,k} + R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,1}| \tag{6.6.15}$$

and substituting in (6.6.13) using equation (6.6.10) gives,

$$|\epsilon^{(j+1)}_{i,k}| = |R^{(j+1)}_{i,k} - R^{(j)}_{i,k} - (R^{(j)}_{i,k} - R^{(j+1)}_{i,k}(1+\epsilon_1)^{-1})(1+\epsilon_2)^{-1}(1+\epsilon_3)^{-1}|$$

$$< 5.01\, \epsilon g , \tag{6.6.16}$$

if $\epsilon < 10^{-3}$.

From (6.6.7) it can be seen that,

$$\varepsilon_{i,k} < 5.01 \quad \varepsilon g(P-k) \qquad (6.6.17)$$

while i=1,2,...N and k=P,P-1,...1, or k=i,i-1,...1,

and from (6.6.9) it can be seen that,

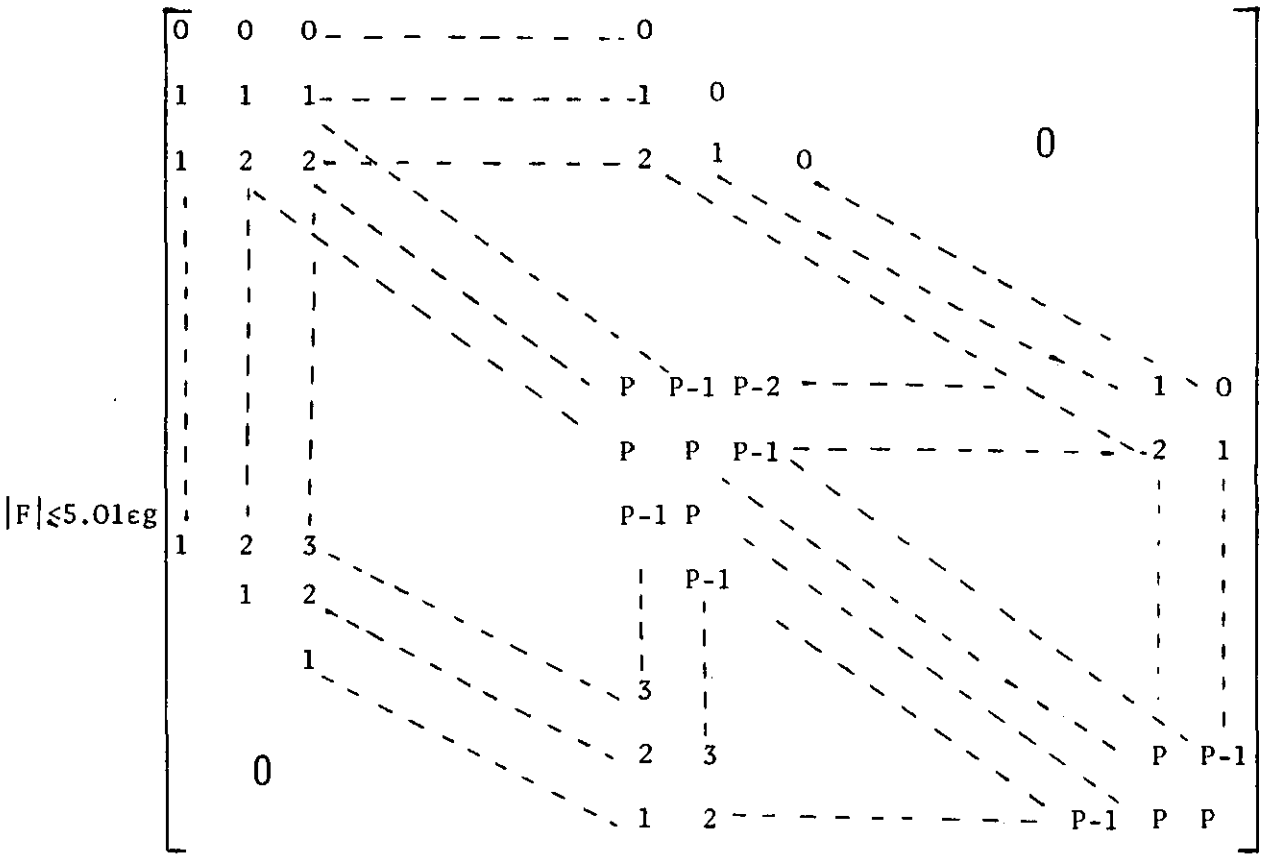
$$\varepsilon'_{i,k} < 5.01 \; \varepsilon g \; (P-k+1) \qquad (6.6.18)$$

while i=2,3,...N and k=P,P-1,....,1 or k=i,i-1,...,1

Therefore the $R_{i,k}$ are calculated exactly for the matrix C+F where the matrix F is defined by,

$$F = \begin{bmatrix} \varepsilon_{1,1} & \varepsilon_{2,2} & \varepsilon_{3,3} & \cdots & \varepsilon_{P,P} & & \\ \varepsilon'_{2,2} & \varepsilon_{2,1} & \varepsilon_{3,2} & \cdots & \varepsilon_{P+1,P} & 0 & \\ \varepsilon'_{3,3} & \varepsilon'_{3,2} & \varepsilon_{3,1} & & & & \varepsilon_{N,P} \\ \vdots & & & & & & \vdots \\ \varepsilon'_{P,P} & & & & & & \varepsilon'_{N,2} \\ & \varepsilon'_{P+1,P} & & & & & \\ & 0 & & \varepsilon'_{N,\bar{P}} & \cdots & \varepsilon'_{N,2} & \varepsilon_{N,1} \end{bmatrix} \qquad (6.6.19)$$

Substituting equations (6.6.17) and (6.6.18) it follows for the matrix F that,



(6.6.20)

The error matrix defined by (6.6.20) is an upper bound only, and in practice, is seldom achieved. The form of the error matrix is different for the various types of band matrix. For example the error matrix for a (14×14) sparse quindiagonal matrix of semi bandwidth 7 of the same form as (5.2.1) has the form

$$F \lesssim 5.01\ \varepsilon g \begin{bmatrix}
0 & 0 & & & & & 0 & & & & & & & \\
1 & 1 & 0 & & & 0 & & 1 & 0 & & & & 0 & \\
 & 1 & 1 & 0 & & & & 1 & 1 & 0 & & & & \\
 & & 1 & 1 & 0 & & & 1 & 1 & 1 & 0 & & & \\
 & & & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & & & \\
 & 0 & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & & \\
1 & 2 & 2 & 2 & 2 & 2 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & \\
 & 1 & 2 & 2 & 2 & 2 & 6 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
 & & 1 & 2 & 2 & 2 & 5 & 6 & 6 & 5 & 4 & 3 & 2 & 1 \\
 & & & 1 & 2 & 2 & 4 & 5 & 6 & 6 & 5 & 4 & 3 & 2 \\
 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 5 & 4 & 3 \\
 & & & & & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 5 & 4 \\
 & & 0 & & & & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 5 \\
 & & & & & & & 1 & 2 & 3 & 4 & 5 & 6 & 6
\end{bmatrix} \qquad (6.6.21)$$

Similarly the bounds on rounding errors can be defined for any of the sparse matrices given in previous chapters.

CHAPTER 7

FURTHER RELATED TOPICS

7.1 INTRODUCTION

In this Chapter is described two pieces of work, that, although complete in themselves indicate areas for further research.

The first section details a method for finding the eigenvalues of a symmetric matrix in ascending or descending order of magnitude, starting from the smallest or largest eigenvalue. This method is then speeded up considerably by the use of Partial Sturm sequences, thus saving the time used in the heavy workload of calculating the full sequence for every iteration.

The subsequent sections describe a modification to the Lanczos method which attempts to extend the range of the method. The Lanczos algorithm transforms a given matrix to tridiagonal form by a similarity transformation and finds the eigenvalues of this matrix. This method suffers from loss in accuracy at the later stages of the transformation on large matrices giving inaccurate eigenvalues. The modified method described tries to counteract the loss in accuracy by transforming the original matrix to a banded matrix, for which the Sturm sequence is now known, instead of to a strictly tridiagonal form. Hence the eigenvalues can be determined to a greater accuracy. Also offered are some ideas for extending the usefulness of the method.

7.2 A METHOD FOR UTILISING PARTIAL STURM SEQUENCES TO FIND THE EIGENVALUES OF A SYMMETRIC MATRIX

The method described here is used to determine the eigenvalues of symmetric sparse quindiagonal matrices of the same form as 5.2.1 by using the secant method to find the zeroes of the determinental equation

$$|C - \lambda I| = 0 \ . \qquad (7.2.1)$$

If $P_N(\lambda)$ is the determinant of the matrix $(C_N-\lambda I)$ and λ^{i-1}, λ^i are two approximations to an eigenvalue then the secant method gives an improved

approximation to the eigenvalue (λ^{i+1}) as,

$$\lambda^{i+1} = \lambda^{i} - \frac{(\lambda^{i}-\lambda^{i-1})P_N(\lambda^{i})}{(P_N(\lambda^{i})-P_N(\lambda^{i-1}))} \qquad (7.2.2)$$

The recursive sequence obtained in section (5.2) for a symmetric sparse quindiagonal matrix gives the relationship

$$R_{i,1}(\lambda) = P_i(\lambda)/P_{i-1}(\lambda), \quad i=1.....N \ . \qquad (7.2.3)$$

The function $R_{N,1}(\lambda)$ therefore has the same zeroes as $P_N(\lambda)$ and can be substituted in (7.2.2) to give,

$$\lambda^{i+1} = \lambda^{i} - \frac{(\lambda^{i}-\lambda^{i-1})R_{N,1}(\lambda^{i})}{(R_{N,1}(\lambda^{i})-R_{N,1}(\lambda^{i-1}))} \qquad (7.2.4)$$

If (7.2.2) is multiplied by $P_{N-1}(\lambda^{i})/P_{N-1}(\lambda^{i})$ it becomes,

$$\lambda^{i+1} = \lambda^{i} - \frac{(\lambda^{i}-\lambda^{i-1})P_N(\lambda^{i})/P_{N-1}(\lambda^{i})}{(P_N(\lambda^{i})/P_{N-1}(\lambda^{i})-P_N(\lambda^{i-1})/P_{N-1}(\lambda^{i}))} \ . \qquad (7.2.5)$$

Now substituting in equation (7.2.5) using (7.2.3) gives the result

$$\lambda^{i+1} = \lambda^{i} - \frac{(\lambda^{i}-\lambda^{i-1})R_{N,1}(\lambda^{i})}{R_{N,1}(\lambda^{i})-R_{N,1}(\lambda^{i-1})P_{N-1}(\lambda^{i-1})/P_{N-1}(\lambda^{i})} \ . \qquad (7.2.6)$$

It can be seen that equations (7.2.6) and (7.2.4) differ only slightly. Obviously both equations converge to zeroes of the determinant of (C-λI), but the convergence rates of the two equations are possibly different.

Certainly it can be seen that close to an eigenvalue after a number of iterations the value of

$$\varepsilon = |\lambda^{i}| - |\lambda^{i-1}| \ , \qquad (7.2.7)$$

will be very small, and therefore the values of $P_{N-1}(\lambda^{i-1})$ and $P_{N-1}(\lambda^{i})$ will be very close. Therefore the factor $P_{N-1}(\lambda^{i-1})/P_{N-1}(\lambda^{i})$ will be

very close to unity. This implies that when a value of λ^i, which, is a close approximation to an eigenvalue, has been attained, that equations (7.2.4) and (7.2.6) have close numerical values and, therefore, the same rates of convergence.

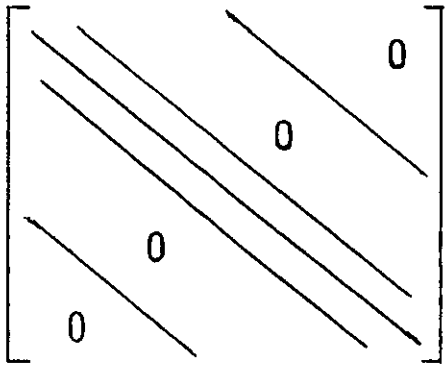
This leaves the question of what effect does the factor $P_{N-1}(\lambda^{i-1})/P_{N-1}(\lambda^i)$ have on the convergence of the method when λ^i is not close to an eigenvalue and equation (7.2.4) is used?

The results of a number of tests under varying conditions indicate that there is negligible difference in timing on a computer between using equation (7.2.4) and (7.2.6) to find the eigenvalues of a matrix with the secant method. It is these results that lead to the formulation of the new sparse secant method. The proof that the two different secant formulae ((7.2.6) and (7.2.4)) converge at very similar rates cannot yet be given. It is intuitively obvious that the convergence rates will be similar, but a further suggestion is given and looked at from a practical viewpoint.

It depends entirely on the starting criteria (which are essentially chosen at random) what the value of λ^{i+1} is, given λ^i, λ^{i-1}. If there are large distances between the values λ^{i-1} to λ^{i+1} any number of eigenvalues could be contained in this range. Thus the method will converge very slowly until a λ^{i+1} is found close to an eigenvalue, when convergence is rapid. So that during this process the ratio $P_{N-1}(\lambda^{i-1})/P_{N-1}(\lambda^i)$ being far removed from 1 does little to slow down a comparatively very slow process, and perhaps even fortuitously places λ^{i+1} close to an eigenvalue thus shortening the search.

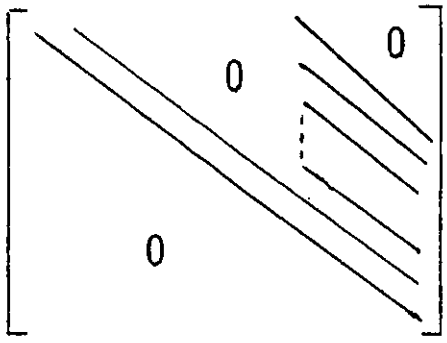
It is this apparent leeway, in the early stages of the method at least, that gave the idea of using a Sturm sequence that was not wholly complete, but simple to calculate, until the proximity of an eigenvalue was achieved. At this point the iteration could be switched to use the correct Sturm sequence to obtain the eigenvalue accurately.

A similar process has already been developed by Evans to solve sets of linear equations which give rise to sparse banded diagonally dominant matrices. If Gaussian elimination is performed on these matrices no interchanges occur. The new elements that are created rapidly diminish to zero along the bands approaching the main diagonal. This is illustrated for a matrix of the same form as (5.2.2),



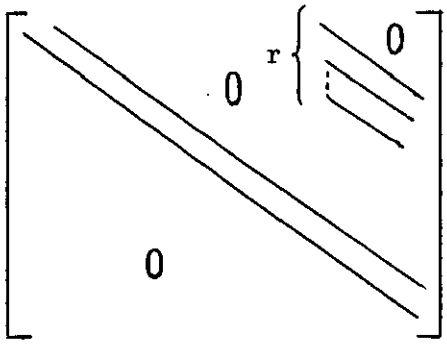
. (7.2.8)

After Gaussian elimination it has the form,



. (7.2.9)

As the elements in the bands closer to the main diagonal diminish to almost zero they can be neglected and not even calculated. The number of bands to be kept to maintain accuracy varies, but is less than 5, and the resultant matrix has the form,



, (7.2.10)

where $r<5$.

This process is used in solving sets of linear equations and the backward and forward substitutions (say) through a matrix of the same form as (7.2.10) are quicker than through a matrix of the same form as (7.2.9). However both types of matrix produce answers to the same accuracy.

It was then considered that if the method described in section (5.2) was used here (i.e., Gaussian elimination without interchanges even on non-diagonally dominant matrices) some terms could be neglected to speed up the calculation of $R_{N,1}(\lambda)$. The matrix that is produced by the method from section (5.2) is,

$$C = \begin{bmatrix} R_{1,1} & R_{2,2} & & & R_{P,P} & & 0 \\ & R_{2,1} & R_{3,2} & 0 & R_{P,P-1} & & \\ & & R_{3,1} & & \vdots & & R_{N,P} \\ & & & & R_{P,2} & & R_{N,P-1} \\ & & 0 & & & & \\ & & & & R_{P,1} & & R_{N,2} \\ & & & & & & R_{N,1} \end{bmatrix} \qquad (7.2.11)$$

If say only FS of the bands were calculated the matrix would have the form,

$$C = \begin{bmatrix} R_{1,1} & R_{2,2} & & & R_{P,P} & & & 0 \\ & R_{2,1} & R_{3,2} & 0 & R_{P,P-1} & R_{P+1,P} & & \\ & & R_{3,1} & & \vdots & & & R_{N,P} \\ & & & & R_{P,P-FS} & & & R_{N,P-1} \\ & & & & & & & \vdots \\ & & & & & & 0 & R_{N,P-FS} \\ & & 0 & & & & & \\ & & & & & & & R_{N,2} \\ & & & & & & & R_{N,1} \end{bmatrix} \qquad (7.2.12)$$

There are obvious savings in time made possible in (7.2.12) by calculating fewer terms, but it is quite obvious that $R_{N,1}$ is inaccurate unless the matrix is diagonally dominant. As the $R_{i,1}$, i=P,P+1,...N are also not accurate bisection cannot be used. This is because one wrong decision during the bisection process due to an incorrect $R_{i,1}$ will prevent even a close approximation being obtained to an eigenvalue. This is why the secant method is used.

The elements of the matrix (7.2.12) are obtained by the method presented in section (5.2) and are defined by the following recursive formula

$$
\left.
\begin{aligned}
&R_{1,1} = C_1-\lambda, \\
&R_{i,2} = b_i \quad , && i=2,3,.....P, \\
&R_{i,1} = C_i-\lambda-R_{i,2}R_{i,2}/R_{i-1,1}, && i=2,3,.....P-1, \\
&R_{i,P} = d_i \quad , && i=P,P+1,....N, \\
&R_{i,k} = -R_{i,k+1}R_{i-P+2,2}/R_{i-P+1,1} \\
&\qquad -(\text{if } i-k+1\geq P: \sum_{j=0}^{FS+1-k} R_{i,P-j}R_{i-k+1,P+1-k-j}/R_{i-P+j+1,i}) \ , \\
&\qquad k=P-1,P-2,.....P-FS, \\
&\qquad i=P,P+1,.....N \quad , \\
&R_{i,2} = b_i - \sum_{j=0}^{FS-1} R_{i,P-j}R_{i-1,P-1-j}/R_{i-P+j+1,1} \ , \\
&\qquad i=P+1,P+2,.....N, \\
&R_{i,1} = C_i-\lambda-R_{i,2}R_{i,2}/R_{i-1,1} - \sum_{j=0}^{FS} R_{i,P-j}R_{i,P-j}/R_{i+P+j+1,1}, \\
&\qquad i=P,P+1,.....N \ .
\end{aligned}
\right\} \qquad (7.2.13)
$$

This sequence was programmed and compared to the full sequence (5.2.22) for large numbers of different matrices with FS varying over the whole possible range (i.e. 2,3,...P-3). The comparisons showed that no strong pattern could be established for the two sequences, no matter how many terms were kept. The only factor noticed was that occasionally the values

of the $R_{i,j}$ of the partial sequence followed quite closely in sign and magnitude those of the full sequence. Although no doubt a detailed statistical analysis of the two sequences will show a strong correlation. It was thought therefore that approximations to eigenvalues could be obtained for little expenditure of time using the partial sequence. Then at some stage later the full Sturm sequence could be used to obtain the eigenvalue correctly. The next step was to find the optimum number of bands to keep (i.e. the size of FS) and the optimum number of iterations to use the partial Sturm sequence on before switching to the full sequence, but first the procedure adopted is described.

The secant method as described by Anderson (1975) is used. First, if say j eigenvalues have been found the matrix must be deflated to avoid re-determining known eigenvalues. This is achieved by dividing $R_{N,1}$ by the sum of the differences of the j known eigenvalues from the current estimate of the $j+1^{th}$ eigenvalue, i.e.

$$R_{N,1} / \sum_{k=1}^{j} (\lambda^{i}_{j+1} - \lambda_k) \quad , \tag{7.2.14}$$

Unfortunately this function has an asymptote on the λ axis. To avoid the method following the asymptote and not converging the numerator is replaced and (7.2.14) becomes,

$$R_{N,1} R_{N-1,1} \cdots\cdots\cdots R_{N-j,1} / \sum_{k=1}^{j} (\lambda^{i}_{j+1} - \lambda_k) \quad , \tag{7.2.15}$$

which, after simple algebraic manipulation, can be written

$$P_N / \sum_{k=1}^{j} (\lambda^{i}_{j+1} - \lambda_k) P_{N-j-1} \quad . \tag{7.2.16}$$

This does not alter earlier statements concerning convergence of the secant method, which were illustrated for the simplest case when no eigenvalue had yet been determined. Function (7.2.15) is more complex than (7.2.4) and alters the factor which distinguishes it from (7.2.6) but in no way alters the analysis or suggestion given.

Anderson (1975) has shown that the eigenvalues of a real symmetric

matrix can be obtained in monotonic ascending or descending order using the secant method. The initial two guesses at an eigenvalue are chosen outside the eigenvalue range, which is determined using Gerschgorin's theorem. The secant method then converges to the largest (smallest) eigenvalue. If the matrix is deflated, and the next two initial approximations to an eigenvalue are made larger (smaller) than the last determined eigenvalue, then the method converges to the largest (smallest) unknown eigenvalue.

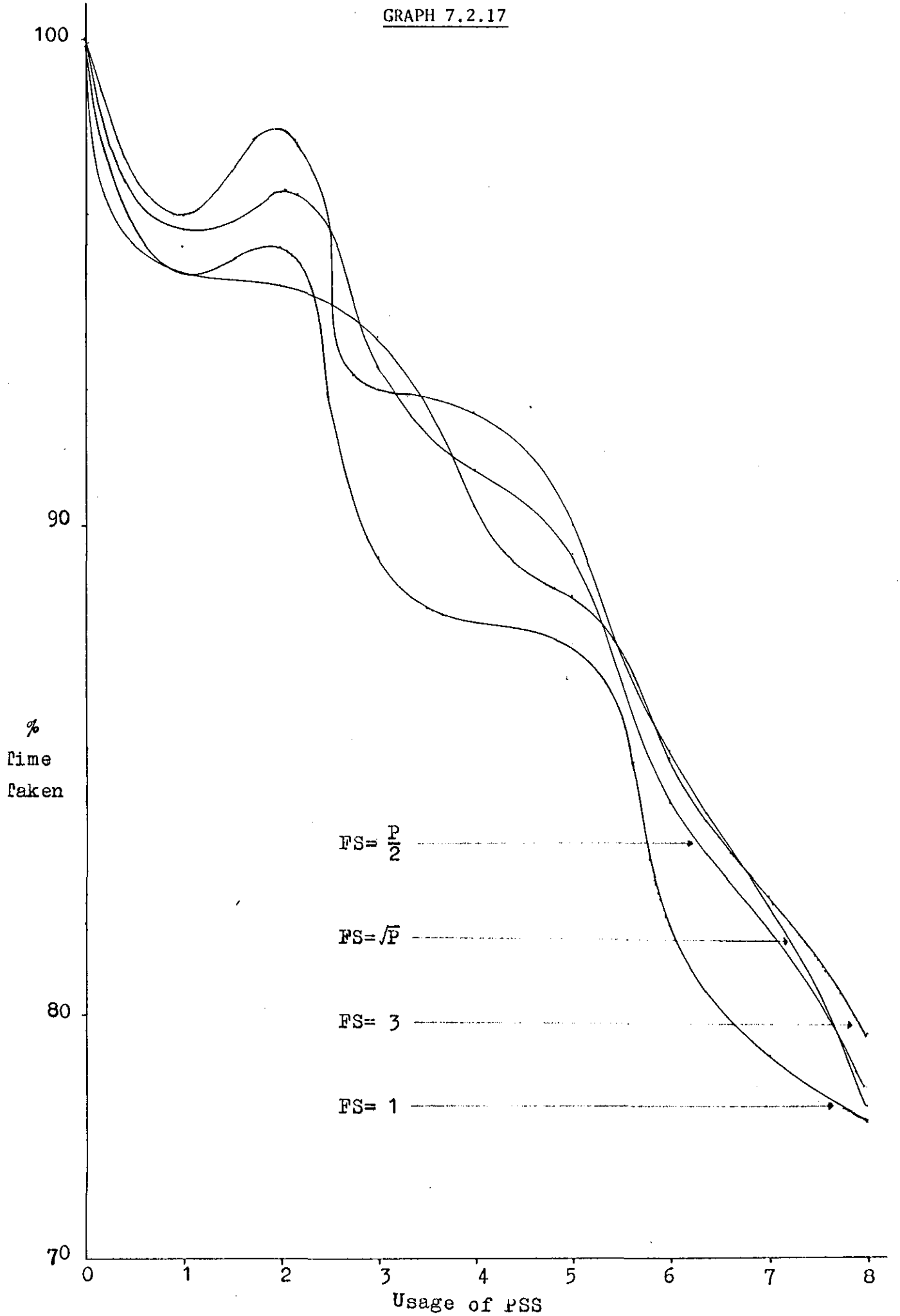
The procedure just described was programmed in ALGOL 60 and is given in Appendix 1 in program 21.

The results for this program are summarised in the graph (7.2.17). Each line represents a different value of FS, or the number of bands retained, and the variable factor is the number of times the partial Sturm sequence is used before reverting to the full Sturm sequence. The results are given as a percentage of the time taken to obtain the results if a full Sturm sequence only was used. The results were obtained for a large number of matrices of different sizes with varying bandwidth. Also varying numbers of eigenvalues were obtained on each occasion, either all, or some of the largest or smallest.

During the large number of tests it was discovered that if the partial sequence was used for more than eight iterations the method did not always determine the eigenvalues in monotonic sequence, and sometimes converged to spurious ones. For this reason the results are only given as far as eight iterations with the partial Sturm sequence.

However the graph indicates that no matter how many terms are kept or for how many attempts, that some improvement in time taken is gained. The results also show that the number of terms kept makes little difference to the time saved. The saving seems to be approximately 20%, and the method is more stable keeping as many terms as possible. When more bands then half the semi-bandwidth are kept (P/2) the time saving

GRAPH 7.2.17


100
90
80
70
% Time Taken
FS= P/2
FS=√P
FS= 3
FS= 1
0
1
2
3
4
5
6
7
8
Usage of PSS

deteriorates, but as many as possible should be kept to ensure that the eigenvalues are determined in the correct order. It is therefore recommended that the number of bands kept be set to half the semi-bandwidth, and the partial sequence be used for eight iterations before reverting to the full sequence.

This method requires further research to fully exploit the possibilites. For example the number of iterations with the partial Sturm sequence can be extended past eight. On the occasions when the correct eigenvalues were obtained in correct sequence, savings of 40% and over were made. Obviously if only the largest 10% (say) of eigenvalues are required then the method is not effective unless it can guarantee the required eigenvalues. However it may be possible to extend the method and reap the benefits. There is also the possibility of using a similar procedure on other types of matrix, or in conjunction with other root finding methods such as Newton-Raphson or even Muller's method.

It is clear, however, that the method in its present form does represent an advance. For example, the method can rival that of bisection. On the occasions where it was considered most efficient to use bisection to find several of the largest or smallest eigenvalues of a symmetric matrix the partial secant method can be used with large savings in time.

Similar procedures to those adopted in (7.2) were applied to the method of inverse iteration used on sparse quindiagonal matrices of the same form as (5.2.1). After considerable programming effort a program was produced that retained only several bands during the elimination or forward substitution stage, despite the difficulties caused by interchanging rows. This program, when test run, confirmed the fact that the method was impracticable. This was mainly because the method was so quick compared to finding the eigenvalues (say), that the effort involved and the space taken with program steps reduced savings to a small percentage (less than 1%). So that the normal inverse iteration procedure

is preferred for finding eigenvectors.

7.3 A MODIFIED LANCZOS METHOD TO DETERMINE THE EIGENVALUES OF A SPARSE QUINDIAGONAL MATRIX

The Lanczos transformation provides an efficient method for determining the eigenvalues and eigenvectors of a small matrix. This work provides some modifications to the method enabling it to be used on larger matrices. To best illustrate the methods used they are only discussed for the simplest case i.e. the real symmetric matrix. The extension to more difficult types of matrix is an easy step conceptually, but entails more work, which tends to hide the ideas being presented. The transformation can break down under certain conditions and steps taken to recover it. Again these are not discussed as they have already been widely investigated and need no further mention here.

The method has been tested on and is essentially intended for use on matrices that are already sparse. This is to provide a quick, easy transformation to a matrix with a more simple Sturm sequence for a rapid solution. The method can equally well be applied to a dense matrix, but it is doubtful that a solution can be obtained quicker or more accurately than the QR or LR methods for this type of matrix. One of the desired features of the method is that no matrix multiplications are needed in the transformation, and thus, if the matrix can be stored in a small number of vectors considerable savings in space are made when the method is programmed on a computer. This is why it is believed the method is most competitive when used on a sparse matrix.

For a detailed analysis of the Lanczos method see Lanczos (1949), Wilkinson (1959), Causey and Gregory (1961), Paige (1971, 1972).

The Lanczos method is briefly described for the transformation

symmetric matrix.

If A is the (N×N) matrix to be transformed, then an arbitrary N-vector, $\underline{b}_1$, is chosen (usually $\underline{b}_1=(1,0,0,\ldots 0)$). Then the vector $\underline{b}_2$ is determined from the following relationships,

where,
$$\left.\begin{aligned} \beta_2\underline{b}_2 &= A\underline{b}_1 - \alpha_1\underline{b}_1 \ , \\ \alpha_1 &= \underline{b}_1 A\underline{b}_1 \end{aligned}\right\} \ , \qquad (7.3.1)$$

and normalising leaves the vector b_2.

The remaining sequence of vectors, b_i, i=3,4,...N+1 are now found using the following relationships,

$$\beta_{i+1}\underline{b}_{i+1} = A\underline{b}_i - \alpha_i\underline{b}_i - \beta_i\underline{b}_{i-1} \ , \quad i=2,3,\ldots N \qquad (7.3.2)$$

where,

and
$$\left.\begin{aligned} \alpha_i &= \underline{b}_i A\underline{b}_i \\ \beta_i &= \underline{b}_{i-1} A\underline{b}_i \end{aligned}\right\} \qquad (7.3.3)$$

As each of the $\beta_i\underline{b}_i$, i=2,3,...N is determined it is normalised leaving the vector $\underline{b}_i$. The vectors $\underline{b}_i$ are also orthogonal (Wilkinson (1959)) and assuming none of these vectors is zero, then $\underline{b}_{N+1}$ must equal zero as it is orthogonal to N non-zero N-vectors. The relationships given by (7.3.1-3) can be written in matrix form as,

$$A(\underline{b}_1\underline{b}_2\ldots\underline{b}_N)=(\underline{b}_1\underline{b}_2\ldots\underline{b}_N)\begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & 0 \\ & \beta_3 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_N \\ 0 & & & \beta_N & \alpha_N \end{bmatrix} \qquad (7.3.4)$$

If the matrix B is the matrix with the b_i as its columns the (7.3.4) is written

$$B^{-1}AB = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & 0 \\ & \beta_3 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_N \\ & 0 & & \beta_N & \alpha_N \end{bmatrix} = C \qquad (7.3.5)$$

Therefore C is a matrix produced by a similarity transform on A, and C and A have the same eigenvalues. Also if x is an eigenvector of C then Bx is an eigenvector of A. The matrix A has now been transformed to a symmetric tridiagonal matrix from which the eigenvalues and eigenvectors can be quickly and easily found using the bisection method and inverse iteration.

The main problem caused by this method when used on a computer is the effect due to rounding errors. In the later stages of the process a calculated $\underline{b}_i$ may be orthogonal to the previous vectors $(\underline{b}_{i-1}, \underline{b}_{i-2})$, but it is no longer orthogonal to $\underline{b}_1$, $\underline{b}_2$, $\underline{b}_3$. A good check on this is that $\underline{b}_{N+1}$ which should be zero, often has appreciable length. This fault has been partially overcome by the introduction of re-orthogonalisation. This is a process by which the current $\underline{b}_i$ being calculated is made orthogonal to all the previously calculated vectors. This vector should already be orthogonal to the previous vectors, but has 'drifted away' due to rounding errors, hence, re-orthogonalisation. It is easily described by the relationship used to obtain $\underline{b}_{i+1}$, which is,

$$\left.\begin{aligned} \underline{b}_{i+1} &= A\underline{b}_i + \alpha_i \underline{b}_i - \beta_i \underline{b}_{i-1} \\ &\quad - \sum_{j=1}^{i-2} \varepsilon_{i,i+1-j} \underline{b}_j \quad , \; i=2,\ldots,N, \\ \text{where} \quad \alpha_i &= \underline{b}_i A \underline{b}_i \; , \\ \beta_i &= \underline{b}_{i-1} A \underline{b}_i \; , \\ \varepsilon_{i,i+1-j} &= \underline{b}_j A \underline{b}_i , \quad j=1,2,\ldots,i-2. \end{aligned}\right\} \qquad (7.3.6)$$

As the vector will be orthogonal to the previous two or three vectors the corrections needed to re-orthogonalise will only be small. The matrix C then becomes,

$$C = \begin{bmatrix} \alpha_1 & \beta_2 & \epsilon_{3,3} & \epsilon_{4,4} & \epsilon_{5,5} & \cdots & \epsilon_{N,N} \\ 1 & \alpha_2 & \beta_3 & \epsilon_{4,3} & \epsilon_{5,4} & \cdots & \epsilon_{N,N-1} \\ & 1 & \alpha_3 & \beta_4 & \epsilon_{5,3} & & \vdots \\ & & \ddots & \ddots & \ddots & & \epsilon_{N,3} \\ & & & \ddots & \ddots & & \beta_N \\ & & 0 & & \ddots & 1 & \alpha_N \end{bmatrix} \qquad (7.3.7)$$

This matrix C, produces more accurate answers, but it is much more difficult to obtain eigenvalues from this matrix than from the tridiagonal matrix (7.3.6).

Some method was desired to retain the orthogonality of the vectors $\underline{b}_i$, i=1,2....N without completely filling in the upper triangle. A compromise solution was proposed so that as each $\underline{b}_i$ i=1,2,.....,N was produced, not only was it made orthogonal to the previous two vectors, but also it was made orthogonal to one other vector ($\underline{b}_N$). This, it was hoped, would substantially reduce the effect of rounding errors by 'pinning' every vector to one of the original pair. The method is as follows.

Two initial vectors were chosen, normalised and mutually orthogonal. Typically they were,

$$\left.\begin{aligned} \underline{b}_1 &= (1,0,0.....0) \\ \underline{b}_N &= (0,0,0.....1) \end{aligned}\right\} , \qquad (7.3.8)$$

Then $\underline{b}_2$ is defined by the equation

$$\beta_2\underline{b}_2 = A\underline{b}_1 - \alpha_1\underline{b}_1 - \beta_1\underline{b}_N \quad , \qquad (7.3.9)$$

where

$$\left.\begin{aligned} \alpha_1 &= \underline{b}_1 A \underline{b}_1 \\ \beta_1 &= \underline{b}_N A \underline{b}_1 \end{aligned}\right\} \quad , \tag{7.3.10}$$

Now when $\beta_2 \underline{b}_2$ is normalised this effectively removes the factor β_2 leaving the vector $\underline{b}_2$.

The vectors $\underline{b}_i$, i=3,4,...N+1 are now determined by the following relationships,

$$\beta_{i+1} \underline{b}_{i+1} = A\underline{b}_i - \alpha_i \underline{b}_i - \beta_i \underline{b}_{i-1}$$

where,

$$\left.\begin{aligned} \alpha_i &= \underline{b}_i A \underline{b}_i \quad , \\ \beta_i &= \underline{b}_{i-1} A \underline{b}_i \quad , \\ \gamma_i &= \underline{b}_N A \underline{b}_i \quad , \end{aligned}\right\} \tag{7.3.11}$$

again β_{i+1} is normalised to remove the factor β_{i+1}.

If B is the matrix with the vectors $\underline{b}_i$, i=1,...N as its columns the transformations can be written as,

$$B^{-1}AB = \begin{bmatrix} \alpha_1 & \beta_2 & & & & \beta_1 \\ \beta_2 & \alpha_2 & \beta_3 & & & \gamma_2 \\ & \beta_3 & \alpha_3 & \ddots & 0 & \gamma_3 \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & \ddots & \beta_N \\ \beta_1 & \gamma_2 & \gamma_3 & \cdots & \beta_N & \alpha_N \end{bmatrix} = C \tag{7.3.12}$$

The matrix C is not periodic tridiagonal, but the Sturm sequence is very similar to that given in Chapter 3. The methods described there can be used to find the eigenvalues of the matrix C efficiently using the bisection method, and also the eigenvectors using inverse iteration.

The Sturm sequence for the matrix C in (7.3.12) is

$$
\left.
\begin{aligned}
q_1(\lambda) &= \alpha_1 - \lambda \quad , \\
s_1(\lambda) &= \beta_1 \\
q_i(\lambda) &= \alpha_i - \lambda - \beta_i^2/q_{i-1}(\lambda), \quad i=2,3,\ldots N-1, \\
s_i(\lambda) &= \gamma_i - s_{i-1}(\lambda)\beta_i/q_{i-1}(\lambda), \quad i=2,3,\ldots N-2, \\
s_{N-1}(\lambda) &= \beta_N - s_{N-2}(\lambda)\beta_{N-1}/q_{N-2}(\lambda) \quad , \\
q_N(\lambda) &= \alpha_N - \sum_{i=1}^{N-1} s_i^2(\lambda)/q_i(\lambda) \quad ,
\end{aligned}
\right\} \qquad (7.3.13)
$$

the $q_i(\lambda)$ then being used in the bisection process.

The program to carry out this method was written in ALGOL 68 and is given in Appendix 1 in program 22.

The results show that this method does give some improvement in the maintenance of the orthogonality of the vectors $\underline{b}_i$, i=1,...N thus improving the results. Typically, with a matrix of order 25 where the answers for the normal Lanczos method are beginning to deteriorate one more figure of accuracy is retained by the modified method. The time taken to perform the transformation for the modified method is not significantly different to that taken for the normal Lanczos algorithm. The bisection algorithm for the modified matrix requires approximately 10% more time than for a tridiagonal matrix.

This method, while not by any means producing a completely satisfactory improvement in performance, does represent a different way of looking at the Lanczos algorithm. If similar methods are pursued further an improved algorithm may be developed.

REFERENCES

ANDERSON N., (1975), *"On computing eigenvalues of matrices with real eigenvalues by the secand method"*,
Royal Inst.Tech. Stockholm, Report, TRITA-NA-7513.

ANDRES T., HOSKINS W.D., McMASTER G.E., (1974), *"A coupled algorithm for the solution of tridiagonal systems"*,
Comp.J., Vol.17, No.4. P.377.

ANDREW A.L., (1973), *"The solution of equations involving centro-symmetric matrices"*,
Technometrics, Vol.15, No.2. P.405.

BARLOW R.H., (1977a), *"Performance of a dual processor parallel processing system"*,
Loughborough Univ. of Tech. Dept. Comp.Studies, Report 43.

BARLOW R.H., (1977b), *"Parallel algorithms for sorting, quadrature, and eigenvalue determination"*,
Loughborough Univ. of Tech. Dept. Comp.Studies, Report 44.

BARTH W., MARTIN R.S., WILKINSON J.H., (1967), *"Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection"*,
Numerische M. Vol.9. P.386.

BJORCK A., GOLUB G.H., (1975), *"Eigenproblems for matrices associated with periodic boundary conditions"*
Linkoping University Report LIH-MAT-R-1974-8.

BOHTE Z., (1974), *"Errors in Gaussian Elimination"*
Publications of the Dept. of Maths. Univ. of Ljubljana, No.6.

BOHTE Z., (1975), *"Bounds for rounding errors in Gaussian elimination for band systems"*,
J.Inst.Maths.Applics., Vol.16, No.2. P. 133.

BUSINGER P.A., (1971), *"Monitoring the numerical stability of Gaussian elimination"*
Numerische Math. Vol.16. P. 360.

CHOW T.S., KOWALK J.S., (1973), *"Sparse matrix problems"*
Internat.J.Numer.Methods Eng., Vol.7. P. 211.

CLASEN R.J., (1966), *"Techniques for automatic tolerance control in linear programming"*
Comm.A.C.M., Vol.9. P. 902.

ERISMAN A., (1973), *"Stability of triangular factorisation of a sparse matrix"*
Numerische Math., Vol.22. P. 183.

EVANS D.J., ATKINSON L.V., (1970), *"An algorithm for the solution of general three term linear systems"*,
Computer J., Vol.13, No.3. P. 323.

EVANS D.J., (1971), *"Numerical solution of the Sturm-Loiuville problem with periodic boundary conditions"*
Conf. on Applics. of Num.Anal., Springer Verlag.

EVANS D.J., (1973), *"An algorithm for the solution of certain systems of linear equations"*,
Computer J., Vol.15, No.4. P. 356.

EVANS D.J., (1974), *"Software for numerical mathematics conference proceedings"*
Academic Press.

EVANS D.J., (1975), *"A recursive algorithm for determining the eigenvalues of a quindiagonal matrix"*,
Computer J., Vol.18, No.1. P. 70.

EVANS D.J., HATZOPOULOS M., (1976), *"The solution of certain banded systems of linear equations using the folding algorithm"*,
Computer J., Vol.19, No.2. P. 184.

FORSYTHE G.E., MOLER C.B., (1967), *"Computer solution of linear algebraic systems"*
Prentice Hall.

GOLUB G.H., ROBERTSON T.N., (1967), *"A generalised Bairstow algorithm"*,
Comm.A.C.M., Vol.10, No.6. P.371.

GOLUB G.H., (1973), *"Some modified matrix eigenvalue problems"*,
SIAM Review, Vol.15, No.2. P. 318.

GREGORY R.T., KARNEY D.L., (1969), *"A collection of matrices for testing computational algorithms"*,
Wiley Interscience.

HATTER D.J., (1973), *"Matrix computer methods of vibration analysis"*
Butterworth.

HILDEBRAND F.B., (1968), *"Finite difference equations and simulations"*,
Prentice-Hall.

HILDEBRAND F.B., (1974), *"Introduction to numerical analysis"*
McGraw-Hill.

HOHN F.E., (1964), *"Elementary matrix algebra"*
Macmillan.

LYNESS, J.N., (1974), *"Computational techniques based on a Lanczos representation"*,
Math.Comp. Vol.28. P. 81.

MARTIN R.S., WILKINSON J.H., (1965), *"Symmetric decomposition of positive definite band matrices"*,
Numerische M., Vol.7. P. 355.

MARTIN R.S., WILKINSON J.H., (1967), *"Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices"*
Numerische M., Vol.9. P. 279.

MARTIN R.S., WILKINSON J.H., (1968), *"Reduction of the symmetric eigenproblem $A\underline{x}=\lambda B\underline{x}$ and related problems to standard form"*,
Numerische M. Vol.11. P. 99.

MILLER K.S., (1964), *Partial differential equations in engineering problems"*
Prentice Hall.

MITCHELL A.P., (1969), *"Computational methods in partial differential equations"*,
Aberdeen University Press.

PAIGE C.C., (1972), *"Computational variants of the Lanczos method for the eigenproblem"*,
J.Inst.Maths.Applics., Vol.10. P. 373.

PETERS G., WILKINSON J.H., (1969), *"Eigenvalues of $A\underline{x}=\lambda B\underline{x}$ with band symmetric A and B"*,
Computer J., Vol.12. P. 398.

PETERS G., WILKINSON J.H., (1970), *"$A\underline{x}=\lambda B\underline{x}$ and the generalised eigenproblem"*,
SIAM J.Num.Anal., Vol.7, No.4. P. 479.

RALL L.B., (Editor), (1975), *"Error in digital computation"*,
Wiley.

REID J.K., (1971), *"A note on the stability of Gaussian elimination"*,
J.Inst.Maths.Applics., Vol.8, P. 374.

REISS E.L., BAUER L., (1972), *"Block five-diagonal matrices and the fast numerical solution of the biharmonic equation"*,
Maths. of Comp. Vol.26,, No.118, P. 311.

NAG LIBRARY MANUAL MK.5.
NAG LTD. OXFORD.

REISS E.L., BAUER L., (1974), *"On the numerical solution of two dimensional elasticity problems"*,
J. of Comp.Phys. Vol.15, No.1, P.21.

ROSE D.J., WILLOUGHBY R.A. (Editors) (1972), *"Sparse matrices and their application"*,
Plenum Press.

SAMEH A.H., KUCK D.J., (1975), *"A parallel QR-algorithm for tridiagonal symmetric matrices"*,
University of Illinois Report.

SCHWARZ H.R., (1968), *"Tridiagonalisation of a symmetric band matrix"*,
Numerische Math., Vol.12. P. 231.

STEWART G.W., (1974), *"Modifying pivot elements in Gaussian elimination"*,
Maths. of Comp. Vol.28, No.126. P. 537.

STEWART G.W., (ROSENFIELD J.L. (ED.)), (1974), *"The numerical treatment of large eigenvalue problems"*,
Proc. of I.F.I.P. Conf. P. 666.

STONE H.S., (1973), *"An efficient parallel algorithm for a tridiagonal linear system"*,
J.Assoc.Comput.Mach., Vol.20, P.27.

STRANG G., FIX G., (1973), *"An analysis of the finite element method"*,
Prentice Hall.

SWEET R.A., (1969), *"A recursive relation for the determinant of a pentadiagonal matrix"*,
Comm. of A.C.M. Vol.12. P.330.

TEWARSON R.P., (1967), *"Row column permutation of sparse matrices"*,
Comp.J. Vol.10. P. 300.

TEWARSON R.P., (1969), *"The Crout reduction for sparse matrices"*,
Comp.J., Vol.12. P.158.

TEWARSON R.P., (1970), *"Computations with sparse matrices"*,
SIAM Review, Vol.12. P. 527.

TEWARSON R.P., (1973), *"Sparse matrices"*,
Academic Press.

TRAUB J.F., (1964), *"Iterative methods for solving functions of a single variable"*,
Prentice Hall.

TRAUB J.F., (1973), *"Complexity of sequential and parallel numerical algorithms"*,
Academic Press.

VARGA R.S., (1962), *"Matrix iterative analysis"*,
Prentice-Hall.

WALSH J., (1966), *"Numerical analysis: An introduction"*,
Academic Press.

WALTMAN W.L., LAMBERT R.J., (1965), *"T-algorithm for tridagonalisation"*, J.SIAM, Vol.13, No.4, P.1065.

WESTLAKE J.R., (1968), *"A handbook of numerical matrix inversion and solution of linear equations"*, John Wiley.

WHITE P.A., (1958), *"The computation of eigenvalues and eigenvectors of a matrix"*, J.SIAM, Vol.6, No.4. P.393.

WILKINSON J.H., (1958a), *"The evaluation of the zeros of ill-conditioned polynomials"*, Numerische M. Vol.1. P.150.

WILKINSON J.H., (1958b), *"The calculation of the eigenvectors of codiagonal matrices"*, Computer J. Vol.1. P. 90.

WILKINSON J.H., (1959), *"The calculation of eigenvectors by the method of Lanczos"*, Computer J. Vol.2. P.148.

WILKINSON J.H., (1960), *"Error analysis of floating point computation"*, Numerische M., Vol.2. P. 319.

WILKINSON J.H., (1961a), *"Error analysis of direct methods of matrix inversion"*, J.A.C.M., Vol.8, P. 281.

WILKINSON J.H., (1961b), *"Rigorous error bounds for computed eigen-systems"*,
Computer J., Vol.4, P. 230.

WILKINSON J.H., (1962), *"Calculation of the eigenvectors of a symmetric tridiagonal matrix by inverse iteration"*,
Numerische M., Vol.4. P. 368.

WILKINSON J.H.,(1963), *"Rounding errors in algebraic processes"*,
H.M.S.O.

WILKINSON J.H.,(1965), *"The algebraic eigenvalue problem"*,
Oxford University Press.

WILKINSON J.H., (1967), *"Two algorithms based on successive linear interpolation"*,
Stanford Univ. Report, CS 60.

WILLOUGHBY R.(Ed.), (1969), *"Proc. Symp. on sparse matrices and their applications"*,
I.B.M. Report RA1(11707).

WILLOUGHBY R., ROSE D.J., (Editors) (1970), *"Sparse matrix applications"*,
Plenum Press,

YANG W.H., LEE E.H., (1974), *"Modal analysis of Floquet waves in composite material"*,
J.App.Mech. Paper 73-APMW-40.

APPENDIX 1

This appendix contains the programs written from the various algorithms described earlier. The programs are mostly written in I.C.L. 1900 ALGOL 60, some however are written in ALGOL 68R and FORTRAN IV as implemented on I.C.L. 1900 machines. The first program is a copy of the program in Numerische Mathematik which is written in the ALGOL 60 reference language as approved by I.F.I.P. and translated to I.C.L. 1900 ALGOL 60.

PROGRAM 1

This program is a copy of the ALGOL program given in Barth et al (1967), it is included for reference. This program is probably the best and most efficient form of a general bisection method and is used in this case in connection with the Sturm sequence for a tridiagonal matrix to determine its eigenvalues. It is possible with little alteration to insert the Sturm sequence for any type of matrix and use the program to determine the eigenvalues of this matrix. Many of the programs to follow are written as modifications to this algorithm.

This program finds the eigenvalues of a tridiagonal matrix as given in (4.2.1). The Sturm sequence for this matrix as obtained by a Laplace expansion is

$$\left.\begin{aligned} P_0(\lambda) &= 1, \quad P_1(\lambda) = c_1-\lambda \quad , \\ P_i(\lambda) &= (c_i-\lambda)P_{i-1}(\lambda)-b_i^2P_{i-2}(\lambda), \ i=2,N \end{aligned}\right\} \qquad (A.1.1)$$

To avoid numerical problems of underflow and overflow this sequence of $P_i(\lambda)$ is replaced by a sequence of $q_i(\lambda)$ where,

$$q_i(\lambda) = P_i(\lambda)/P_{i-1}(\lambda), \ i=1;N \qquad (A.1.2)$$

and the sequence of (A.1.1) becomes,

$$\left.\begin{aligned} q_1(\lambda) &= c_1-\lambda \quad , \\ q_i(\lambda) &= c_i-\lambda-b_i^2/q_{i-1}(\lambda), \ i=2,N \end{aligned}\right\} \qquad (A.1.3)$$

It is the sequence (A.1.3) that is used in the program, and it is this section that can be replaced by the Sturm sequence from other matrices.

```
'PRØCEDURE' BISECT(C,B,BETA,N,M1,M2,EPS1,RELFEH,EPS2,Z,X);
'VALUE' N,M1,M2,EPS1,RELFEH;
'REAL' EPS1,EPS2,RELFEH;
'INTEGER' N,M1,M2,Z;
'ARRAY' C,B,X,BETA;
'ØOMMENT' C is the diagonal, B the sub-diagonal and BETA the squared sub-
          diagonal of a symmetric tridiagonal matrix of order N. The
          eigenvalues LAMBDA[M1],.....LAMBDA[M2], where M2 is not less
          than M1 and LAMBDA[I+1] is not less than LAMBDA[I], are
          calculated by the method of bisection and stored in vector X.
          Bisection is continued until the upper and lower bounds for
          an eigenvalue differ by less than EPS1 unless at some stage,
          the upper and lower bounds differ only in the least significant
          digits. EPS2 gives an extreme upper bound for the error in any
          eigenvalue, but for certain types of matrices the small eigen-
          values are determined to a very much higher accuracy. In this
          case, EPS1 should be set equal to the error to be tolerated in
          the smallest eigenvalue. It must not be set equal to zero;
'BEGIN'
     'REAL' H,XMIN,XMAX;
     'INTEGER' I;
     'CØMMENT' calculation of XMIN,XMAX, maximum and minimum values of
               eigenvalue range
     BETA[1]←B[1]←0;
     XMIN←C[N]-ABS(B[N]);
     XMAX←C[N]+ABS(B[N]);
     'FØR' I←N-1 'STEP'-1'UNTIL' 1 'DØ'
     'BEGIN'
          H←ABS(B[I])+ABS(B[I+1]);
          'IF' C[I]+H 'GT' XMAX 'THEN' XMAX←C[I]+H;
          'IF' C[I]-H 'LT' XMIN 'THEN' XMIN←C[I]-H;
     'END';
     EPS2←RELFEH*('IF' XMIN+XMAX 'GT' 0 'THEN'
          XMAX 'ELSE' - XMIN);
     'IF' EPS1 'LE' 0 'THEN' EPS1←EPS2;
     EPS2←0.5*EPS1+7*EPS2;
     'CØMMENT' inner block;
     'BEGIN'
          'INTEGER' A,K;
          'REAL' Q,X1,XU,X0;
          'ARRAY' WU[M1:M2];
          X0←XMAX;
          'FØR' I←M1 'STEP' 1 'UNTIL' M2 'DØ'
          'BEGIN'
               X[I]←XMAX;
               WU[I]←XMIN;
          'END';
          Z←0;
          'CØMMENT'loop for the k^th eigenvalue;
          'FØR' K←M2 'STEP' -1 'UNTIL' M1 'DØ'
          'BEGIN'
               XU←XMIN;
               'FØR' I←K 'STEP -1 'UNTIL' M1 'DØ'
               'BEGIN'
                    'IF' XU 'LT' WU[I] 'THEN'
                    'BEGIN'
                         XU←WU[I];
                         'GØTØ' CØNTIN;
                    'END';
```

```
                    'END'
CØNTIN;             'IF' XO 'GT' X[K] 'THEN' XO←X[K];
                    'FØR' X1←(XU+XO)/2 'WHILE'
                         XO-XU 'GT'2*RELFEH*(ABS(XU)+ABS(XO))+EPS1
                         'DØ'
                    'BEGIN'
                         Z←Z+1;
                         'COMMENT' this section to the next comment is the
                         section that can readily be replaced by any
                         appropriate Sturm sequence;
                         A←0;
                         Q←1;
                         'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
                         'BEGIN'
                              Q←C[I]-X1-('IF' Q 'NE' 0 'THEN' BETA[I]/Q
                                   'ELSE ABS(B[I])/RELFEH);
                              'IF' Q<0 'THEN' A←A+1;
                         'END';
                         'CØMMENT' end of Sturm sequence;
                         'IF' A 'LT' K 'THEN'
                         'BEGIN'
                              'IF' A 'LT' M1 'THEN' XU←WU[M1]←X1
                              'ELSE'
                              'BEGIN'
                                   XU←WU[A+1]←X1;
                                   'IF' X[A] 'GT' X1 'THEN' X[A]←X1
                              'END';
                         'END'
                         'ELSE' XO←X1;
                    'END';
                    X[K]←(XO+XU)/2;
          'END';
     'END';
'END';
```

PROGRAM 2

This program finds the eigenvalues of a symmetric periodic tridiagonal matrix by bisection. It is written as a modification to Program 1.

```
'PRØCEDURE' PBISECT(C,B,BETA,N,M1,M2,EPS1,RELFEH,EPS2,Z,X);
'VALUE' N,M1,M2,EPS1,RELFEH;
'REAL' EPS1,EPS2,RELFEH;
'INTEGER' N,M1,M2,Z;
'ARRAY' C,B,X,BETA;
'CØMMENT' C is the diagonal, B the sub-diagonal, BETA the squared sub-
     sub-diagonal, and B[1] the corner element of a symmetric periodic
     tridiagonal matrix of order N. The eigenvalues LAMBDA[M1],.....,
     LAMBDA[M2], where M2 is not less than M1 and LAMBDA[I+1] is not
     less than LAMBDA[I] are calculated by the method of bisection and
     stored in vector X. Bisection is continued until the upper and
     lower bounds for an eigenvalue differ by less than EPS1 unless at
     some stage, the upper and lower bounds differ only in the least
     significant digits. EPS2 gives an extreme upper bound for the
     error in any eigenvalue, but for certain types of matrices the
     small eigenvalues are determined to a very much higher accuracy.
     In this case EPS1 should be set equal to the error to be tolerated
     in the smallest eigenvalue. It must not be set equal to zero;
'BEGIN'
     'REAL' H,XMIN,XMAX;
     'INTEGER' I;
     'CØMMENT' calculation of XMIN, XMAX, maximum and minimum values of
          eigenvalue range
     H←ABS(B[N])+ABS(B[1]);
     XMIN←C[N]-H;
     XMAX←C[N]+H;
     'FØR' I←N-1 'STEP -1 'UNTIL' 1 'DØ'
     'BEGIN'
          H←ABS(B[I])+ABS(B[I+1]);
          'IF' C[I]+H 'GT' XMAX 'THEN' XMAX←C[I]+H;
          'IF' C[I]-H 'LT' XMIN 'THEN' XMIN←C[I]-H;
     'END';
     EPS2←RELFEH*('IF'XMIN+XMAX 'GT' 0 'THEN' XMAX 'ELSE'-XMIN);
     'IF' EPS1 'LE' 0 'THEN' EPS1←EPS2;
     EPS2←0.5*EPS1+7*EPS2;
     'CØMMENT' inner block;
     'BEGIN'
          'INTEGER'A,K;
          'REAL' Q, X1,XU,XO;RT,Y,U;
          'ARRAY' WU[M1:M2];
          XO←XMAX;
          'FØR' I←M1 'STEP' 1 'UNTIL' M2 'DØ'
          'BEGIN'
               X[I]←XMAX;
               WU[I]←XMIN;
          'END';
          Z←0;
          'CØMMENT' loop for the kth eigenvalue;
          'FØR' K←M2 'STEP' -1 'UNTIL' M1 'DØ'
          'BEGIN'
               XU←XMIN;
               'FØR' I←K 'STEP' -1 'UNTIL' M1 'DØ'
```

```
                'BEGIN'
                     'IF' XU 'LT' WU[I] 'THEN'
                     'BEGIN'
                          XU←WU[I];
                          'GØTØ' CØNTIN;
                     'END';
                'END';
CØNTIN:         'IF' XO 'GT' X[K] 'THEN' XO←X[K];
                'FØR' X1←(XU+XO)/2 'WHILE' XO-XU 'GT'
                     2*RELFEH*(ABS(XU)+ABS(XO))+EPS1 'DØ'
                'BEGIN'
                     Z←Z+1;
                     'CØMMENT' section to compute the Sturm sequence;
                     A←0;
                     Q←C[1]-X1;
                     'IF' Q 'EQ' 0 'THEN' Q←ABS(B[I])*RELFEH;
                     'IF' Q 'LT' 0 'THEN' A←A+1;
                     RT←B[1];
                     Y←C[N]-X1;
                     'FØR' I←2 'STEP' 1 'UNTIL' N 'DØ'
                     'BEGIN'
                          Y←Y-RT*RT/Q;
                          RT←-RT*B[I]/Q;
                          Q←C[I]-X1-BETA[I]/Q;
                          'IF' Q 'EQ' 0 'THEN' Q←ABS(B[I])*RELFEH;
                          'IF' Q 'LT' 0 'THEN' A←A+1;
                     'END';
                     RT←RT+B[N];
                     Q←Y-RT*RT/Q;
                     'IF' Q 'LT' 0 'THEN' A←A+1;
                     'CØMMENT' end of Sturm sequence;
                     'IF' A 'LT' K 'THEN'
                     'BEGIN'
                          'IF' A 'LT' M1 'THEN' XU←WU[M1]←X1
                          'ELSE'
                          'BEGIN'
                               XU←WU[A+1]←X1;
                               'IF' X[A] 'GT' X1 'THEN' X[A]←X1;
                          'END';
                     'END'
                     'ELSE' XO←X1;
                'END';
                X[K]←(XO+XU)/2;
           'END';
     'END';
'END';
```

PROGRAM 3

. This program finds the eigenvalues of an unsymmetric periodic tridiagonal matrix of the form of (3.4.6), using Bairstow's method.

```
'PRØCEDURE' BAIRS(C,B,A,AD,N,EPS,EIG,EIGI);
'VALUE' C,B,A,AD,N,EPS;
'ARRAY' C,B,EIG,EIGI;
'REAL' A,AD,EPS;
'INTEGER' N;
'CØMMENT' C is the main diagonal, B the sub-diagonal and A,AD are the
     unsymmetric corner elements. N is the order of the matrix, EPS
     is set for the required number of figures of accuracy. The results
     are given in EIG, EIGI;
'BEGIN'
     'ARRAY'BN,GN,AN,L,M,V,W,E,F,R,X[0:N];
     'REAL' J,K,ALF,BET,JC,KC,BT,AAD;
     'INTEGER' I,IN,Z,LE;
     LE←Z←0;
     BT←1;
     'FØR' I←2 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          BT←-BT*B[I];
          BN[I]←B[I]*B[I];
          GN[I]←AN[I]←0;
     'END';
     AN[0]←GN[0]←AN[1]←GN[1]←0;
     BT←BT*(A+AD);
     AAD←A*AD;
     J←K←1;
     I←0;
LAB1:L[I]←V[I]←W[I]←V[I+1]←E[I]←E[I+1]←F[I]←F[I+1]←R[I]←R[I+1]←X[I]←X[I+1]←0
     L[I+1]←-1;
     M[I]←W[I+1]←1;
     M[I+1]←C[I+1]-AN[I];
     Z←Z+1;
     'FØR' IN←I+2 'STEP'1 'UNTIL' N-1 'DØ'
     'BEGIN'
          L[IN]←(C[IN-J])*L[IN-1]-L[IN-2]*BN[IN]-M[IN-1];
          M[IN]←C[IN]*M[IN-1]-K*L[IN-1]-BN[IN]*M[IN-2]-AN[IN-1];
          V[IN]←(C[IN]-J)*V[IN-1]-BN[IN]*V[IN-2]-N[IN-1];
          W[IN]←C[IN]*W[IN-1]-K*V[IN-1]-BN[IN]*W[IN-2]-GN[I-1];
          E[IN]←(C[IN]-J)*E[IN-1]-F[IN-1]-BN[IN]*E[IN-2];
          F[IN]←C[IN]*F[IN-1]-K*E[IN-1]-L[IN-1]-BN[IN]*F[IN-2];
          R[IN]←(C[IN]-J)*R[IN-1]-X[IN-1]-BN[IN]*R[IN-2]
          X[IN]←C[IN]*X[IN-1]-K*R[IN-1]-V[IN-1]-BN[IN]*X[IN-2];
     'END';
     L[N]←(C[N]-J)*L[N-1]-M[N-1]-BN[N]*L[N-2]-AAD*V[N-1];
     M[N]←C[N]*M[N-1]-K*L[N-1]-BN[N]*M(N-2]-AN[N-1]-AAD*W[N-1];
     M[N]←M[N]+J*L[N];
     E[N]←(C[N-J])*E[N-1]-F[N-1]-BN[N]*E[N-2]-AAD*R[N-1];
     F[N]←C[N]*F[N-1]-L[N-1]-K*E[N-1]-BN[N]*F[N-2]-AAD*X[N-1];
```

```
     F[N]←F[N]+J*E[N];
     ALF←K*E[N]+J*F[N];
     BET←F[N]*F[N]-ALF*E[N];
     JC←(E[N]*M[N]-F[N]*L[N])/BET;
     KC←(ALF*L[N]-F[N]*M[N])/BET;
     K←K+KC;
     J←J+JC;
     'IF' LE 'EQ' 1 'THEN'
     'BEGIN'
          LE←0;
          I←I+1;
          JC←J*J+4*K;
          'IF' JC 'LT' 0 'THEN'
          'BEGIN'
               JC←-JC;
               EIGI[I]←SQRT(JC)/2;
               EIGI[I+1]←-EIGI[I];
               EIG[I]← EIG[I+1]←J/2;
          'END'
          'ELSE'
          'BEGIN'
               JC←SQRT(JC)/2;
               EIG[I]←J/2-JC;
               EIG[I+1]←J/2+JC;
               EIGI[I]←EIGI[I+1]←0;
          'END';
          I←I+1;
          'IF' N-I 'LE' 1 'THEN' 'GOTO' LAB3;
          'FØR' IN←I-2 'STEP' 1 'UNTIL' N-1 'DØ'
          'BEGIN'
               AN[IN]←L[IN];
               GN[IN]←V[IN];
          'END';
          AN[N]←L[N];
          Z←0;
          J←K←C[I];
          'GØTØ' LAB1;
     'END';
     'IF' ABS(JC) 'LT' EPS 'AND' ABS(KC) 'LT' EPS 'THEN'
     LE←1;
     'GØTØ' LAB1;
LAB3:'IF' N-I 'EQ' 1 'THEN'
     'BEGIN'
          EIG[N]←C[N]-L[N-1];
          EIGI[N]←0;
     'END';
'END';
```

PROGRAM 4

This procedure finds all the eigenvalues of a symmetric periodic tridiagonal matrix by Newtons method.

```
'PRØCEDURE' NEWSTURM (C,B,N,EPS,QZR,EIG);
'CØMMENT' C is the main diagonal and B[2],....B[N] is the sub-diagonal.
     The element B[1] is the corner element.  N is the order of the
     matrix and EPS the desired accuracy.  QZR is the limit of machine
     accuracy used to replace zero elements.  The eigenvalues are stored
     in EIG;
'VALUE' C,B,N,EPS,QZR;
'ARRAY' C,B,EIG;
'INTEGER' N;
'REAL' EPS,QZR;
'BEGIN'
     'ARRAY' BN;
     'REAL' X,Y,Z,MA,MI,LAMBDA,BT,RT,XD,RTD;
     'INTEGER' I,L;
     'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
     BN[I]←B[I]*B[I];
     LAMBDA←1;
     L←1;
LABA:'IF' L 'EQ' N+1 'GØTØ' EXIT;
LABB:X←C[1]-LAMBDA;
     'IF'ABS(X) 'EQ' 0 'THEN X←QZR;
     XD←Z←-1;
     RTD←0;
     MA←XD/X;
     RT←B[1];
     Y←C[N]-LAMBDA;
     'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
     'BEGIN'
          MI←X*X;
          Z←Z-(Z*X*RT*RTD-XD*RT*RT)/MI;
          Y←Y-RT*RT/X;
          RTD←-(RTD*X-XD*RT)*B[I]/MI;
          XD←-1+BN[I]*XD/MI;
          RT←-RT*B[I]/X;
          X←C[I]-LAMBDA-BN[I]/X;
          'IF' ABS(X) 'EQ' 0 'THEN' X←QZR;
          MA←MA+XD/X;
     'END';
     MI←X*X;
     Z←Z+(BN[N]*XD/MI-2*(RTD*X-XD*RT)*B[N]-2*RT*RTD*X-XD*RT*RT)/MI;
     RT←RT*BN[N];
     X←Y-RT*RT/X;
     MA←MA+Z/X;
     'IF' 1 'GE' 2 'THEN'
     'FØR' I←1 'STEP' 1 'UNTIL' L-1 'DØ'
     MAX←MAX-1/(LAMBDA-EIG[I]);
     MAX←1/MAX;
     LAMBDA←LAMBDA-MAX;
     'IF'ABS(MAX/LAMBDA) 'LT' EPS 'ØR' ABS(LAMBDA) 'LT' EPS
     'THEN'
     'BEGIN'
          EIG[L]←LAMBDA;
          LAMBDA←LAMBDA+1;
```

```
            L←L+1;
            'GØTØ' LABA;
        'END';
        'GØTØ' LABB;
 EXIT:
'END';
```

PROGRAM 5

This procedure determines all the eigenvalues of an unsymmetric periodic tridiagonal matrix using Newton's method. The program is written in ALGOL-68R to take advantage of the facilities for handling complex arithmetic.

```
'PROC' TRINEWT=([]'REAL'C,B,D,'REAL' EPS,QZR,'INT' N,'REF'[]'CØMPLEX' EIG):
'BEGIN'
'COMMENT'  C is the main diagonal, B[2],....B[N] and D[2],.....,D[N] are the
           two sub-diagonals.  The elements B[1] and D[1] are the corner
           elements.  N is the order of the matrix and EPS is input as the
           desired accuracy.  QZR is the limit of machine accuracy and is
           used to replace elements close to zero.  The eigenvalues are
           stored in the complex array EIG. 'COMMENT';
     'CØMPLEX' P,Q,X,Y,Z,MAX,MIN,LAMBDA,BT,RT,XD,RTD;
     [1:N]'CØMPLEX'BN;
     'INT' L←1;
     'REAL' ZER←0.1↑40;
     LAMBDA←1.123;
LABA:(L=N+1!'GØTØ'EXIT);
LABB:X←C[1]-LAMBDA;
     ('ABS'(X)<ZER!X←QZR);
     P←0;
     Q←D[1];
     RTD←0;
     RT←B[1];
     XD←-1;
     Z←-1;
     MAX←XD/X;
     Y←C[N]-LAMBDA;
     'FØR' I 'FRØM' 2 'TØ' N-1 'DØ'
     'BEGIN'
          MIN←X*X;
          Z←Z-(X*(P*RT+RTD*Q)-XD*Q*RT)/MIN;
          Y←Y-RT*Q/X;
          P←-(P*X-XD*Q)*D[I]/MIN;
          Q←-Q*D[I]/X;
          RTD←(RTD*X-XD*RT)*B[I]/MIN;
          XD←-1+BN[I]/X;
          RT←-RT*B[I]/X;
          X←C[I]-LAMBDA-BN[I]/X;
          ('ABS'(X)<ZER!X←QZR);
          MAX←MAX+XD/X
     'END';
     MIN←X*X;
     Z←Z+(BN[N]*XD-B[N]*(X*P-Q*XD)-D[N]*(X*RTD-RT*XD)-X*(Q*RTD+P*RT)-XD*Q*RT)/MIN;
     Q←Q+D[N];
     RT←RT+B[N];
     X←Y-RT*Q/X;
     ('ABS'(X)<ZER!X←QZR);
     MAX←MAX Z/X;
     (L>=2!'FØR' I 'TØ' L-1 'DØ' MAX←MAX-1/(LAMBDA-EIG[I]));
     MAX←1/MAX;
     LAMBDA←LAMBDA-MAX;
```

```
      'IF' 'ABS'(MAX/LAMBDA)<EPS 'ØR' 'ABS'(LAMBDA)<EPS 'THEN'
      'BEGIN'
           EIG[L]←LAMBDA;
           LAMBDA←2,987?1.0;
           L←L+1;
           'GØTØ' LABA
      'END'
      'ELSE'
      'GØTØ' LABB
      'FI'
 EXIT:
'END';
```

PROGRAM 6

This procedure determines the eigenvalues of a symmetric centro-symmetric matrix by bisection. The diagonal and sub-diagonals are only recorded as far as the centre elements +2, as the matrix is centro-symmetric.

```
'PRØCEDURE' CENTRØBISECT(C,B,BETA,N,M1,M2,EPS1,RELFEH,EPS2,Z,X);
'VALUE' N,M1,M2,EPS1,RELFEH;
'REAL' EPS1,EPS2,RELFEH;
'INTEGER' N,M1,M2,Z;
'ARRAY' C,B,X,BETA;
'COMMENT' C,B, and D are the first N/2+2 elements of the diagonal,
    sub-diagonal, and squared sub-diagonal of symmetric, centro-
    symmetric tridiagonal matrix of order N. The eigenvalues LAMBDA[M1],
    .....,LAMBDA[M2], where M2 is not less than M1 and LAMBDA[I+1] is not
    less than LAMBDA[I+2], are calculated by the method of bisection and
    stored in vector X. Bisection is continued until the upper and lower
    bounds for an eigenvalue differ by less than EPS1 unless, at some
    stage, the upper and lower bounds differ only in the least significant
    digits. EPS2 gives an extreme upper bound for the error in any
    eigenvalue, but for certain types of matrices the small eigenvalues
    are determined to a much higher accuracy. In this case, EPS should
    be set equal to the error to be tolerated in the smallest eigenvalue.
    It must not be set equal to zero.
'BEGIN'
    'REAL H,XMIN,XMAX;
    'INTEGER' I,N2;
    'COMMENT' calculation of XMIN,XMAX, maximum and minimum values of
        eigenvalue range
    N2←N'/'2;
    XMIN←C[1]-ABS(B[2]);
    XMAX←C[1]+ABS(B[2]);
    'FØR' I←N2 'STEP' -1 'UNTIL' 2 'DØ'
    'BEGIN'
        H←ABS(B[I])+ABS(B[I+1];
        'IF' C[I]+H 'GT' MAX 'THEN' XMAX←C[I]+H;
        'IF' C[I]-H 'LT' MIN 'THEN' XMIN←C[I]-H;
    'END';
    EPS2←RELFEH*('IF'XMIN+XMAX 'GT' 0 'THEN' XMAX 'ELSE'XMIN);
    'IF' EPS1 'LE' 0 'THEN' EPS1←EPS2;
    EPS2←0.5*EPS1+7*EPS2;
    'CØMMENT' inner block;
    'BEGIN'
        'INTEGER' A,K;
        'REAL' Q,X1,XU,XO;
        'ARRAY' WU[M1:M2];
        XO←XMAX;
        'FØR' I←M1 'STEP' 1 'UNTIL' M2 'DØ'
```

```
          'BEGIN'
               X[I]←XMAX;
               WU[I]←XMIN;
          'END';
          Z←0;
          'CØMMENT' Loop for the kth eigenvalue;
          'FØR' K←M2 'STEP' -1 'UNTIL' M1 'DØ'
          'BEGIN'
               XU←XMIN;
               'FØR I←K 'STEP' -1 'UNTIL' M1 'DØ'
               'BEGIN'
                    'IF' XU 'LT' WU[I] 'THEN'
                    'BEGIN'
                       XU←WU[I];
                       'GØTØ' CØNTIN;
                    'END';
               'END';
CØNTIN:        'IF'XO 'GT' X[K] 'THEN' XO←X[K];
               'FOR X1←(XU+XO)*0.5 'WHILE' XO-XU 'GT'
               2*RELFEH*(ABS(XU)+ABS(XO))+EPS 'DØ'
               'BEGIN'
                    Z←Z+1;
                    'CØMMENT' Sturm sequence;
                    A←0; Q←1;
                    'FØR' I←1'STEP' 1 'UNTIL' N2 'DØ'
                    'BEGIN'
                       Q←C[I]-X1-('IF' Q 'NE' 0 'THEN'
                       BETA[I]/Q'ELSE'ABS(B[I]/RELFEH);
                       'IF' Q 'LT' 0 'THEN' A←A+2;
                    'END';
                    'IF' N2*2-N'EQ'0'THEN'
                    'BEGIN'
                       Q←Q-('IF'Q'NE'0'THEN'BETA[N2+1]/Q
                       'ELSE' ABS(B[N2+1]/RELFEH);
                       'IF' Q 'LT' 0 'THEN' A←A+1;
                    'END';
                    'CØMMENT' End of Sturm sequence;
                    'IF' A 'LT' K 'THEN'
                    'BEGIN'
                       'IF' A 'LT' M1 'THEN' XU←WU[M1]←X1
                       'ELSE'
                       'BEGIN'
                            XU←WU[A+1]←X1;
                            'IF X[A] 'GT' X1 'THEN' X[A]←X1
                       'END';
                    'END'
                    'ELSE' XO←X1;
               'END';
               X[K]←(XO+XU)/2;
          'END';
     'END';
'END';
```

PROGRAM 7

This procedure determines the eigenvectors of a centro-symmetric matrix using a modified inverse iteration method.

```
'PRØCEDURE' VECTOR(C,B,N,M,EIG,EIGVEC);
'CØMMENT' C is the diagonal and B the sub-diagonal of the tri-diagonal
    matrix of order N.  Only N '/'2+2 of the elements of C and N'/'2+1
    elements of B are used.  M is the number of vectors that are required.
    The eigenvalues are supplied in EIG and the results are placed in the
    rows of EIGVEC;
'VALUE' C,B,N,M,EIG;
'ARRAY' C,B,EIG,EIGVEC;
'INTEGER' N,M;
'BEGIN'
    'ARRAY' AN,BN,CN,D,S[1:(N+4)'/'2];
    'INTEGER' I,H,L;
    'REAL' V;
    'INTEGER''ARRAY' IC[1:(N+4)'/'2];
    'CØMMENT' Gaussian elimination;
    'FØR' L←1 'STEP' 1 'UNTIL' M 'DØ'
    'BEGIN' H←N'/'2;
        CN[1]←C[1]-EIG[L]; IC[1]←0;
        'FØR' I←2 'STEP' 1 'UNTIL' H+2 'DØ'
        'BEGIN'
            D[I ]←0; IC[I]←0;
            CN[I]←C[I]-EIG[L];
            BN[I]←AN[I]←B[I];
        'END';
        'FØR' I←1 'STEP' 1 'UNTIL' H-1 'DØ'
        'BEGIN'
            'IF' ABS(CN[I]) 'LT' ABS(AN[I+1]) 'THEN'
            'BEGIN'
                V←CN[I]; CN[I]←AN[I+1]; AN[I+1]←V;
                V←BN[I+1]; BN[I+1]←CN[I+1]; CN[I+1]←V;
                V←D[I+2]; D[I+2]←BN[I+2]; BN[I+2]←V;
                IC[I]←1;
            'END';
            AN[I]←AN[I+1]/CN[I];
            CN[I+1]←CN[I+1]-BN[I+1]*AN[I];
            BN[I+2]←BN[I+2]-D[I+2]*AN[I];
        'END';
        'CØMMENT' Section to eliminate the centre elements depending on
            whether N is odd or even;
        'IF' H*2 'EQ' N 'THEN'
        'BEGIN' 'CØMMENT' N even;
            'IF' ABS(CN[H]) 'LT' ABS(BN[H+1]) 'THEN'
            'BEGIN'
                V←BN[H+1]; AN[H+1]←BN[H+1]←CN[H];
                CN[H]←CN[H+1]←V; IC[H]←1;
            'END'
            'ELSE'
```

```
     'BEGIN'
          CN[H+1]←CN[H]; AN[H+1]←BN[H+1];
     'END';
     AN[H]←AN[H+1]/CN[H];
     CN[H+1]←CN[H+1]-BN[H+1]*AN[H];
'END'
'ELSE'
'BEGIN' 'CØMMENT' N odd;
     BN[H+2]←AN[H+1]; AN[H+2]←BN[H+1];
     CN[H+2]←CN[H];
     'BEGIN'
          V←CN[H]; CN[H]←AN[H+1]; AN[H+1]←V;
          V←BN[H+1]; BN[H+1]←CN[H+1]; CN[H+1]←V;
          V←D[H+2]; D[H+2]←BN[H+2]; BN[H+2]←V;
          IC[H]←1;
     'END';
     AN[H]←AN[H+1]/CN[H];
     CN[H+1]←CN[H+1]-AN[H]*BN[H+1];
     BN[H+2]←BN[H+2]-AN[H]*D[H+2];
     'IF' ABS(CN[H+1]) 'LT' ABS(AN[H+2]) 'THEN'
     'BEGIN'
          V←CN[H+1]; CN[H+1]←AN[H+2]; AN[H+2]←V;
          V←BN[H+2]; BN[H+2]←CN[H+2]; CN[H+2]←V;
          IC[H+1]←1;
     'END';
     AN[H+1]←AN[H+2]/CN[H+1];
     CN[H+2]←CN[H+2]-AN[H+1]*BN[H+2];
'END';
H←(N-1)'/'2;
'FØR'I←1 'STEP' 1 'UNTIL' H+2 'DØ'
S[I]←1;
'IF' CN[H+2] 'EQ' 0 'THEN' CN[H+2]←2↑(-37);
'CØMMENT' Closest number to machine zero if matrix decomposes.
     Back substitution now takes place;
S[H+2]←S[H+2]/CN[H+2];
S[H+1]←(S[H+1]-S[H+2]*BN[H+2])/CN[H+1];
'FØR' I←H 'STEP' -1 'UNTIL' 1 'DØ'
S[I]←(S[I]-S[I+1]*BN[I+1]-S[I+2]*D[I+2])/CN[I];
'CØMMENT' The forward substitution with stored interchanges and
     elimination factors is performed;
'FØR' I←1 'STEP' 1 'UNTIL' H+1 'DØ'
'BEGIN'
     'IF' IC[I] 'EQ' 1 'THEN'
     'BEGIN'
          V←S[I]; S[I]←S[I+1]; S[I+1]←V;
     'END';
     S[I+1]←S[I+1]-S[I]*AN[I];
'END';
'CØMMENT' Back substitution is now performed for the second time;
S[H+2]←S[H+2]/CN[H+2];
S[H+1]←(S[H+1]-S[H+2]*BN[H+2])/CN[H+1];
'FØR' I←H 'STEP' -1 'UNTIL' 1 'DØ'
S[I]←(S[I]-S[I+1]*BN[I+1]-S[I+2]*D[I+2])/CN[I];
'CØMMENT' Vector is now normalised;
```

```
          V←S[H+1]*S[H+1];
          'IF (H+1)*2 'NE' N 'THEN' V←V/2;
          'FØR I←1 'STEP 1 ' UNTIL' H 'DØ'
          V←V+S[I]*S[I];
          V←SQRT(V*2);
          'FØR' I←1 'STEP' 1 'UNTIL' H+2 'DØ'
          S[I]←S[I]/V;
          'CØMMENT' The full eigenvector is now written in EIGVEC;
          V←S[H+2]*S[H+1-(H+1)*2+N];
          'IF' V 'LT' N 'THEN'
          'BEGIN'
               'FØR' I←1 'STEP' 1 'UNTIL' H 'DØ'
               'BEGIN'
                    EIGVEC[L,I]←S[I];
                    EIGVEC[L,N+1-I]←-S[I];
               'END';
          'END'
          'ELSE'
          'FØR' I←1 'STEP' 1 'UNTIL' H 'DØ'
          EIGVEC[L,I]←EIGVEC[L,N+1-I]←S[I];
          'IF' (H+1)*2 'EQ' N 'THEN'
          'BEGIN'
               EIGVEC[L,H+1]←S[H+1];
               EIGVEC[L,H+2]←S[H+2];
          'END'
          'ELSE'
          EIGVEC[L,H+1]←S[H+1];
     'END';
'END';
```

PROGRAM 8

This program finds the eigenvalues of a symmetric tridiagonal matrix using parallel processing on a bisection algorithm. The program is written in ALGOL 60 with the addition of the FORK and JOIN parallel processing constructs. These statements are self explanatory and indicate where parallel processing is performed and where it ends.

```
'PRØCEDURE' PTRIBIS(C,B,BETA,N,M1,M2,EPS1,RELFEH,EPS2,Z,X);
'VALUE' N,M1,M2,EPS1,RELFEH;
'ARRAY' C,B,X,BETA;
'REAL' EPS1,EPS2,RELFEH;
'INTEGER' N,M1,M2,Z;
'CØMMENT' C is the diagonal, B the sub-diagonal and BETA the squared
     sub-diagonal of a symmetric tridiagonal matrix of order N. The
     eigenvalues LAMBDA[M1],....LAMBDA[M2], where M2 is not less than
     M1 and LAMBDA[I+1] is not less than LAMBDA[I], are calculated by
     the method of bisection and stored in the vector X. Bisection is
     continued until the upper and lower bounds for an eigenvalue differ
     by less than EPS1 unless at some earlier stage, the upper and lower
     bounds for an eigenvalue differ only in the least significant digits.
     EPS2 gives an extreme upper bound for the error in any eigenvalue but
     for certain types of matrices the small eigenvalues are determined to
     a very much higher accuracy. In this case, EPS1 should be set equal
     to the error to be tolerated in the smallest eigenvalue. It must
     not be set equal to zero;
'BEGIN'
     'REAL' H,XMIN,XMAX;
     'INTEGER' I;
     'CØMMENT' Calculation of minimum and maximum eigenvalue range;
     BETA[1]←B[1]←0;
     XMIN←C[N]-ABS(B[N]);
     XMAX←C[N]+ABS(B[N]);
     'FØR' I←N-1 'STEP' -1 'UNTIL' 1 'DØ'
     'BEGIN'
          H←ABS(B[I])+ABS(B[I+1]);
          'IF' C[I]+H 'GT' XMAX 'THEN' XMAX←C[I]+H;
          'IF' C[I]-H 'LT' XMIN 'THEN' XMIN←C[I]-H;
     'END';
     EPS2←RELFEH*('IF' XMIN+XMAX 'GT' 0 'THEN' XMAX 'ELSE' -XMIN);
     'IF' EPS1 'LE' 0 'THEN' EPS1←EPS2;
     EPS2←0.5*EPS1+7*EPS2;
     'CØMMENT' Inner block;
     'BEGIN'
          'ARRAY' WU[M1:M2];
          'INTEGER' A,AU,J,K,N2;
          'REAL' Q,QU,X1,XU,XO;
          XO←XMAX;
          'FØR' I←M1 'STEP' 1 'UNTIL' M2 'DØ'
          'BEGIN'
```

```
                X[I]←XMAX;
                WU[I]←XMIN;
           'END';
           Z←0;
           N2←N'/'2;
           'COMMENT' Loop for the kth eigenvalue;
           'FOR K←M2 'STEP' -1 'UNTIL' M1 'DO'
           'BEGIN'
                XU←XMIN;
                'FOR' I←K 'STEP' -1 'UNTIL' M1 'DO'
                'BEGIN'
                     'IF' XU 'LT' WU[I] 'THEN'
                     'BEGIN'
                          XU←WU[I];
                          GOTO CONTIN;
                     'END';
                'END';
CONTIN:         'IF' XO 'GT' X[K] 'THEN' XO←X[K];
                'FOR' X1←(XU+XO)/2 'WHILE' XO-XU 'GT' 2*RELFEH
                     *(ABS(XU)+ABS(XO))+EPS1 'DO'
                'BEGIN'
                     Z←Z+1;
                     'COMMENT'  STURM SEQUENCE;
                     AU←A←0;
                     QU←Q←1;
                     'FORK' L1,L2;
L1:                  'BEGIN'
                          'FOR' I←1 'STEP' 1 'UNTIL N2 'DO'
                          'BEGIN'
                               Q←C[I]-X1-('IF' Q
                               'NE' 0 'THEN' BETA[I]/Q
                               'ELSE' ABS(B[I]/RELFEH));
                               'IF' Q 'LT' 0 'THEN' A←A+1
                          'END';
                     'END';
                     'GOTO' L3;
L2:                  'BEGIN'
                          'FOR' J←N 'STEP' -1 'UNTIL' N-N2+1 'DO'
                          'BEGIN'
                               QU←C[J]-X1-('IF' QU 'NE'
                               0 'THEN' BETA[J+1]/Q 'ELSE'
                               ABS(B[J+1]/RELFEH));
                               'IF' QU 'LT' 0 'THEN' AU←AU+1;
                          'END';
                     'END';
L3:                  'JOIN' L1,L2:
                     A←A+AU;
                     'IF' N2*2-N 'EQ' 0 'THEN'
                     'BEGIN'
                          Q←QU-('IF' Q 'NE' 0 'THEN' BETA[N2+1]/Q
                          'ELSE' ABS(B[N2+1]/RELFEH));
                     'END'
                     'ELSE'
                     'BEGIN'
                          'IF' QU 'LT' 0 'THEN' A←A+1;
```

```
                    Q←C[N2+1]-X1-('IF'Q'NE'0'THEN'
                    BETA[N2+1]/Q 'ELSE' ABS(B[N2+1]/RELFEH)
                    -('IF' QU 'NE' 0 'THEN' BETA[N2+2]/QU
                    'ELSE' ABS(B[N2+2]/RELFEH));
               'END';
               'IF' Q 'LT' 0 'THEN' A←A+1;
               'IF' A.'LT' K 'THEN'
               'BEGIN'
                    'IF' A 'LT' M1 'THEN'
                    XU←WU[M1]←X1;
                    'ELSE'
                    'BEGIN'
                         XU←WU[A+1]←X1;
                         'IF X[A] 'GT' X 'THEN' X[A]←X1;
                    'END';
               'END'
               'ELSE'
               XO←X1
          'END';
          X[K]←(XO+XU)/2;
     'END'
'END'
'END';
```

PROGRAM 9

This program finds the eigenvalues of a symmetric tridiagonal matrix using the bisection method. The method is modified by a version of the folding algorithm (Evans and Hatzopoulos) to enable the program to be run in a parallel fashion using two processors. The program is written in standard FORTRAN with the addition of FORK and JOIN routines to allow parallel processing (Barlow 1977a, 1977b) as implemented on the Loughborough University of Technology dual processor Interdata 70.

```
      SUBRØUTINE STURMP(C,BB,EIG,NB,EPS,QZRB)
C     The main diagonal of the matrix is stored in C, the sub-diagonal
C     in BB.  NB is the order of the matrix and EPS the accuracy required.
C     QZRB is a number close to machine zero, used to replace any zero
C     divisiors that occur in the calculation of the Sturm sequence.
C     The resulting eigenvalues are placed in EIG.
      DIMENSIØN C(81),BB(81),EIG(81),CN(81)
      CØMMON/CU/CN,B(81),KXC,KYC,XC,YC,N,M
      N=NB
C     The eigenvalue bounds are now calculated using Gerschgorins theorem
      ANEW=ABS(BB(N))
      AMAX=C(N)+ANEW
      AMIN=C(N)-ANEW
      BB(1)=0
      NMIN1=N-1
      DØ 101 I=1,NMIN1
      ANEW=ABS(BBI))+ABS(BB(I+1))
      ANEWER=C(I)+ANEW
      ANEW=C(I)-ANEW
      IF(ANEWER.GT.AMAX)AMAX=ANEWER
      IF(ANEW.LT.AMIN)AMIN=ANEW
101   CØNTINUE
      AMIN=AMIN-0.1
      AMAX=AMAX+0.1
      DØ 102 I=2,N
102   B(I)=BB(I)*BB(I)
      M=N/2
      L=0
501   IF (L.EQ.N) GØTØ 500
C     Beginning of main loop to find Lth eigenvalue
      ALAMBA=(AMIN+AMAX)*0.5
502   DØ 103 I=1,N
103   CN(I)=C(I)-ALAMBA
      $FØRK 1,2;3
C     Entering parallel mode to determine upper and lower halves of
C     Sturm sequence simultaneously
```

```
1       CØNTINUE
        KX=0
        IF(CN(1)) 5003,504,503
504     X=QZRB
        GØTØ 505
5003    KX=1
503     X=CN(1)
505     DØ 104 I=2,M
        X=CN(I)-B(I)/X
        IF(X.EQ.0.0)X=QZRB
        IF(X.LT.0.0)KX=KX+1
104     CØNTINUE
        XC=X
        KXC=KX
        GØTØ 3
2       CØNTINUE
        KY=0
        IF(CN(N)) 5006,507,506
507     Y=QZRB
        GØTØ 508
5006    KY=1
506     Y=CN(N)
508     NMIN1=N-1
        NMIN2=N-M+1
        JST=2*N-M
        DØ 105 I=NMIN2,NMIN1
        J=JST-I
        Y=CN(J)-B(J+1)/Y
        IF(Y.EQ.0.0)Y=QZRB
        IF(Y.LT.0.0)KY=KY+1
105     CØNTINUE
        YC=Y
        KYC=KY
3       $JØIN
        K=KYC+KXC
        IF(M*2-N) 509,510,509
509     X=CN(M+1)-B(M+1)/XC-B(M+2)/YC
        GØTØ 511
510     IF(YC.LT.0.0)K=K-1
        X=YC-B(M+1)/XC
511     IF(X.LT.0.0)K=K+1
        IF(L-K)512,513,512
513     ANEWER=AMIN
        AMIN=ALAMBA
        ALAMBA=ALAMBA+(ALAMBA-ANEWER)*0.5
        GØTØ 502
512     ALAMBA=(ALAMBA+AMIN)*0.5
        IF((ALAMBA-AMIN)*0.5.GE.EPS)GØTØ 502
        AMIN=2.0*ALAMBA-AMIN
        LPL1=L+1
        DO 107 I=LPL1,K
107     EIG(I)=ALAMBA
        L=K
        GØTØ 501
500     CØNTINUE
        RETURN
        END
```

PROGRAM 10

This program finds the eigenvectors of a symmetric tridiagonal matrix by inverse iteration using two processors. The program is written in standard I.C.L. ALGOL 60 except for the introduction of 'fork' and 'join' statements which indicate where parallel processing can be performed.

```
'PRØCEDURE' PARVEC(C,B,N,M,EIG,EIGVEC);
'CØMMENT' C is the diagonal and B the sub-diagonal of the tridiagonal
     matrix of order N. M is the number of vectors required, and the
     number of eigenvalues supplied in EIG. The resulting vectors are
     placed in the rows of EIGVEC;
'VALUE' C,B,N,M,EIG;
'ARRAY' C,B,EIG,EIGVEC;
'INTEGER' N,M;
'BEGIN'
     'ARRAY' AN,CN,BN,D,S[1:N];
     'INTEGER' I,H,L;
     'REAL' V;
     'INTEGER' 'ARRAY' IC[1:N];
     'FØR' L←1 'STEP' 1 'UNTIL' M 'DØ'
     'CØMMENT' Loop for each eigenvector;
     'BEGIN'
          H←N'/'2;
          'CØMMENT' Set up work vectors with copy of matrix;
          CN[1]←C[1]-EIG[L]; IC[1]←0;
          'FØR' I←2 'STEP' 1 'UNTIL' N 'DØ'
          'BEGIN'
               D[I]←IC[I]←0;
               CN[I]←C[I]-EIG[L];
               BN[I]←AN[I]←B[I];
          'END';
          'CØMMENT' Elimination or forward substitution;
          'FØRK' L1,L2;
          'CØMMENT' Upper half of matrix;
L1:       'FØR' I←N 'STEP' -1 'UNTIL' N-H+2 'DØ'
          'BEGIN'
               'IF ABS(CN[I]) 'LT' ABS(BN[I]) 'THEN'
               'BEGIN'  'CØMMENT' Interchange if necessary;
                    V←CN[I]; CN[I]←BN[I]; BN[I]←V;
                    V←AN[I];AN[I]←CN[I-1]; CN[I-1]←V;
                    V←D[I]; D[I]←AN[I-1]; AN[I-1]←V;
                    IC[I]←1;
               'END';
               BN[I]←BN[I]/CN[I];
               CN[I-1]←CN[I-1]-BN[I]*AN[I];
               AN[I-1]←AN[I-1]-BN[I]*D[I];
          'END';
          'GØTØ' L3;
L2:       'CØMMENT' Lower half of matrix;
          'FØR' I←1 'STEP' 1 'UNTIL'H-1 'DØ'
          'BEGIN'
              'IF' ABS(CN[I]) 'LT' ABS(AN[I+1]) 'THEN'
```

```
          'BEGIN'
               V←CN[I]; CN[I]←AN[I+1]; AN[I+1]←V;
               V←BN[I+1];  BN[I+1]←CN[I+1]; CN[I+1]←V;
               V←D[I+2]; D[I+2]←BN[I+2]; BN[I+2]←V;
               IC[I]←1;
          'END';
          AN[I]←AN[I+1]/CN[I];
          CN[I+1]←CN[I+1]-BN[I+1]*AN[I];
          BN[I+2]←BN[I+2]-D[I+2]*AN[I];
     'END';
L3:  'JØIN' L2,L3;
     'CØMMENT' The centre elements must be eliminated, and the
          method used depends on whether the matrix is odd or
          even.  This cannot be done in parallel;
     'IF' H*2 'EQ' N 'THEN'
     'BEGIN' 'CØMMENT' Matrix is even;
          'IF' ABS(CN[H]) 'LT' ABS(AN[H+1]) 'THEN'
          'BEGIN'
               V←CN[H]; CN[H]←AN[H+1]; AN[H+1]←V;
               V←BN[H+1]; BN[H+1]←CN[H+1]; CN[H+1]←V;
               IC[H]←1;
          'END';
          AN[H]←AN[H+1]/CN[H];
          CN[H+1]←CN[H+1]-BN[H+1]*AN[H];
     'END'
     'ELSE'
     'BEGIN' 'CØMMENT' Matrix is order odd;
          'IF' ABS(CN[H]) 'LT' ABS(AN[H+1]) 'THEN'
          'BEGIN'
               V←CN[H]; CN[H]←AN[H+1]; AN[H+1]←V;
               V←BN[H+1]; BN[H+1]←CN[H+1]; CN[H+1]←V;
               V←D[H+2]; D[H+2]←BN[H+2];  BN[H+2]←V;
               IC[H]←1;
          'END';
          AN[H]←AN[H+1]/CN[H];
          CN[H+1]←CN[H+1]-AN[H]*BN[H+1];
          BN[H+2]←BN[H+2]-AN[H]*D[H+2];
          'IF' ABS(CN[H+1]) 'LT' ABS(AN[H+2]) 'THEN'
          'BEGIN'
               V←CN[H+1]; CN[H+1]←AN[H+2]; AN[H+2]←V;
               V←BN[H+2]; BN[H+2]←CN[H+2]; CN[H+2]←V;
               IC[H+1]←1;
          'END';
          AN[H+1]←AN[H+2]/CN[H+1];
          CN[H+2]←CN[H+2]-AN[H+1]*BN[H+2];
     'END';
     H←(N-1)'/'2;
     'CØMMENT' Set the initial value of the eigenvector to all 1's;
     'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ' S[I]←1;
     'IF' CN[H+2] 'EQ' 0 'THEN' CN[H+2]←QZR;
     'CØMMENT' QZR is the smallest number on the machine, not zero,
          for the decomposing case.  Then back substitution is
          performed for the centre elements.
     S[H+2]←S[H+2]/CN[H+2];
     S[H+1]←(S[H+1]-S[H+2]*BN[H+2])/CN[H+1];
     'CØMMENT' Back substitution is carried out through both halves
          of the matrix, at the same time;
     'FØRK' L4,L5;
L4:  'FØR' I←H+3 'STEP' 1 'UNTIL' N 'DØ'
```

```
          S[I]←(S[I]-S[I+1]*AN[I]-S[I-2]*D[I])/CN[I];
          'GØTØ' L6;
L5:       'FØR' I←H 'STEP' -1 'UNTIL' 1 'DØ'
          S[I]←(S[I]-S[I+1]*BN[I+1]-S[I+2]*D[I+2])/CN[I];
L6:       'JØIN' L4,L5;
          'CØMMENT' Forward substitution or elimination for the second
               time.  Performed on the eigenvector only using stored
               interchanges and elimination factors;
          'FØRK' L7,L8;
L7:       'FØR' I←1 'STEP' 1 'UNTIL' N'/'2-1 'DØ'
          'BEGIN'
               'IF' IC[I] 'EQ' 1 'THEN'
               'BEGIN'
                    V←S[I]; S[I]←S[I+1]; S[I+1]←V;
               'END';
               S[I+1]←S[I+1]-S[I]*AN[I];
          'END';
          'GØTØ'L9;
L8:       'FØR' I←N 'STEP' -1 'UNTIL' N-N'/'2+2 'DØ'
          'BEGIN'
               'IF' IC[I] 'EQ' 1 'THEN'
               'BEGIN'
                    V←S[I]; S[I]←S[I-1]; S[I-1]←V;
               'END';
               S[I-1]←S[I-1]-S[I]*BN[I];
          'END';
L9:       'JØIN' L7,L8;
          'CØMMENT' Eliminate through the centre elements;
          'FØR' I←N'/'2 'STEP' 1 'UNTIL' N-N'/'2 'DØ'
          'BEGIN'
               'IF' IC[I] 'EQ' 1 'THEN'
               'BEGIN'
                    V←S[I]; S[I]←S[I+1]; S[I+1]←V;
               'END';
               S[I+1]←S[I+1]-S[I]*AN[I];
          'END';
          S[H+2]←S[H+2]/CN[H+2];
          S[H+1]←(S[H+1]-S[H+2]*BN[H+2])/CN[H+1];
          'FØRK' L10,L11;
L10:      'FØR' I←H+3 'STEP' 1 'UNTIL' N 'DØ'
          S[I]←(S[I]-S[I-1]*AN[I]-S[I-2]*D[I])/CN[I];
          GØTØ L12;
L11:      'FØR' I←H 'STEP' -1 'UNTIL' 1 'DØ'
          S[I]←(S[I]-S[I+1]*BN[I+1]-S[I+2]*D[I+2])/CN[I];
L12:      'JØIN' L10,L11;
          V←S[N]*S[N];
          'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
          V←V+S[I]*S[I];
          V←1/SQRT(V);
          'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
          EIGVEC[L,I]←S[I]*V;
     'END';
'END';
```

PROGRAM 11

This program finds the eigenvectors of a symmetric tridiagonal matrix by inverse iteration using two processors. The program is written in standard FORTRAN with the addition of FORK and JOIN routines as implemented on the Loughborough University of Technology dual processor Interdata 70.

```
      SUBRØUTINE EIGVEP(C,B,NM,EIG,EIGVEC)
C     The main diagonal is stored in C, the sub-diagonal in B, the order
C     of matrix in NM, and the eigenvalue for which the eigenvector is
C     required in EIG.  The resulting eigenvector is normalised and placed
C     in EIGVEC.
      DIMENSIØN C(81),B(81),EIGVEC(81),AN(81),BN(81)
     1 CN(81),D(81),S(81),IC(81)
      CØMMØN/C3/C(81),B(81),EIGVEC(81)
      CØMMØN/C1/AN(81),BN(81),CN(81),D(81),S(81),IC(81)
      CØMMØN/C2/N
C     Setting up the variables in common store
      N=NM
      IH=N/2
      CN(1)=C(1)-EIG
      IC(1)=0
      DØ 102 I=2,N
      D(I)=0
      IC(I)=0
      CN(I)=C(I)-EIG
      BN(I)=B(I)
      AN(I)=B(I)
102   CØNTINUE
      $FØRK 1020,1030;1040
C     Program now forks to perform Gaussian elimination from both ends of
C     the main diagonal of the matrix simultaneously
1020  CØNTINUE
      NST=N-IH+2
      I=N+1
      DØ 103 I1=NST,N
      I=I-1
      IF(ABS(CN(I)).GE.ABS(BN(I))GØTØ 500
      V=CN(I)
      CN(I)=BN(I)
      BN(I)=V
      V=AN(I)
      AN(I)=CN(I-1)
      CN(I-1)=V
      V=D(I)
      D(I)=AN(I-1)
      AN(I-1)=V
      IC(I)=1
500   BN(I)=BN(I)/CN(I)
      CN(I-1)=CN(I-1)-BN(I)*AN(I)
      AN(I-1)=AN(I-1)-BN(I)*D(I)
```

```
103     CØNTINUE
        GØTØ 1040
1030    CØNTINUE
        JFIN=IH-1
        DØ 104 J=1,JFIN
        IF(ABS(CN(J)).GE.ABS(AN(J+1))) GØTØ 501
        W=CN(J)
        CN(J)=AN(J+1)
        AN(J+1)=W
        W=BN(J+1)
        BN(J+1)=CN(J+1)
        CN(J+1)=W
        W=D(J+2)
        D(J+2)=BN(J+2)
        BN(J+2)=W
        IC(J)=1
501     AN(J)=AN(J+1)/CN(J)
        CN(J+1)=CN(J+1)-BN(J+1)*AN(J)
        BN(J+2)=BN(J+2)-D(J+2)*AN(J)
104     CØNTINUE
1040    $JØIN
C       Elimination of the centre elements is performed seperately to
C       allow for N being odd or even
        IF(IH*2.NE.N) GØTØ 502
        IF(ABS(CN(IH)).GE.ABS(AN(IH+1))) GØTØ 503
        V=CN(IH)
        CN(IH)=AN(I+1)
        AN(IH+1)=V
        V=BN(IH+1)
        BN(IH+1)=CN(IH+1)
        CN(IH+1)=V
        IC(IH+1)=1
503     AN(IH)=AN(IH+1)/CN(IH+1)
        CN(IH+1)=CN(IH+1)-BN(IH+1)*AN(IH)
        GØTØ 509
502     IF(ABS(CN(IH)).GE.ABS(AN(IH+1))) GOTO 505
        V=CN(IH)
        CN(IH)=AN(IH+1)
        AN(IH+1)=V
        V=BN(IH+1)
        BN(IH+1)=CN(IH+1)
        CN(IH+1)=V
        V=D(IH+2)
        D(IH+2)=BN(IH+2)
        BN(IH+2)=V
        IC(IH)=1
504     AN(IH)=AN(IH+1)/CN(IH)
        CN(IH+1)=CN(IH+1)-AN(IH)*BN(IH+1)
        BN(IH+2)=BN(IH+2)-AN(IH)*D(IH+2)
        IF(ABS(CN(IH+1)).GE.ABS(AN(IH+2))) GØTØ 505
        V=CN(IH+1)
        CN(IH+1)=AN(IH+2)
        AN(IH+2)=V
        V=BN(IH+2)
        BN(IH+2)=CN(IH+2)
```

```
      CN(IH+2)=V
      IC(IH+1)=1
505   AN(IH+1)=AN(IH+2)/CN(IH+1)
      CN(IH+2)=CN(IH+2)-AN(IH+1)*BN(IH+2)
509   IH=(N-1)/2
      DØ 105 I=1,N
105   S(I)=1
C     Replace zero in the decompsing case
      IF(CN(IH+2).EQ.0.0) CN(IH+2)=0.0000001
C     Back substitution is now performed through the three non zero vectors
C     of the remaining two triangular half matrices (CN,BN,D).
      S(IH+2)=S(IH+2)/CN(IH+1)
      S(IH+1)=(S(IH+1)-S(IH+2)*BN(IH+2))/CN(IH+1)
      $FORK 1060,1070; 1080
C     Back substitution is now performed through the remaining half
C     matrices simultaneously.
1060  CØNTINUE
      IST=IH+3
      DØ  106 I=IST,N
106   S(I)=(S(I)-S(I-1)*AN(I)-S(I-2)*D(I))/CN(I)
      GØTØ 1080
1070  CØNTINUE
      DØ 107 J1=1,IH
      J=IH+1-J1
107   S(J)=(S(J)-S(J+1)*BN(J+1)-S(J+2)*D(J+2))/CN(J)
1080  $JØIN
      $FORK 1081,1091;1099
C     A second forward substitution or Gaussian elimination is now
C     performed on the eigenvector (S) using only stored elimination
C     factors and interchanges, again in parallel mode.
1081  CØNTINUE
      IFIN=N/2-1
      DØ 108 I=1,IFIN
      IF(IC(I).NE.1) GØTØ 506
      V=S(I)
      S(I)=S(I+1)
      S(I)1)=V
506   S(I+1)=S(I+1)-S(I)*AN(I)
108   CØNTINUE
      GØTØ 1099
1091  CØNTINUE
      JIS=N-N/2+2
      DØ 109 J1=JIS,N
      I=N+N-N/2+2-J1
      IF(IC(I).NE.1)GØTØ 507
      W=S(I)
      S(I)=S(I-1)
      S(I-1)=W
507   S(I-1)=S(I-1)-S(I)*BN(I)
109   CØNTINUE
1099  $JØIN
      IST=N/2
      IFIN=N-IST
      DØ 110 I=IST,IFIN
      IF (IC(I).NE.1) GØTØ 508
```

```
      V=S(I)
      S(I)=S(I+1)
      S(I+1)=V
508   S(I+1)=S(I+1)-S(I)*AN(I)
110   CØNTINUE
      S(IH+2)=S(IH+2)/CN(IH+2)
      S(IH+1)=(S(IH+1)-S(IH+2)*BN(IH+2))/CN(IH+1)
       FORK 1111, 1121; 1129
C     Back substitution is performed a second time in parallel mode.
1111  CØNTINUE
      IST=IH+3
      DØ 111 I=IST,N
      S(I)=(S(I)-S(I-1)*AN(I)-S(I-2)*D(I))/CN(I)
111   CØNTINUE
      GØTØ 1129
1121  CØNTINUE
      DØ 112 J1=1,IH
      I=IH+1-J1
      S(I)=(S(I)-S(I+1)*BN(I+1)-S(I+2)*D(I+2))/CN(I)
112   CØNTINUE
1129   JØIN
      V=S(1)*S(1)
C     The eigenvector is now normalised
      DØ 113 I=2,N
      V=V+S(I)*S(I)
113   CØNTINUE
      V=SQRT(V)
      DØ 114 I=1,N
      EIGVEC(I)=S(I)/V
114   CØNTINUE
      RETURN
      END
```

PROGRAM 12

This procedure uses the Sturm sequence given in section 5.2 and a bisection method to obtain the eigenvalues of a sparse symmetric quin-diagonal matrix. The program is written as a modification of PROGRAM 1.

```
'PRØCEDURE' STURM(C,B,D,N,M1,M2,P,EPS1,RELFEH,EPS2,Z,X);
'CØMMENT' C is the diagonal, B the sub-diagonal, and D the diagonal of
   semi-bandwidth P of the matrix of order N (N>4). The eigenvalues M1
   to M2 (M1≤M2, eigenvalue 1 being the smallest) are calculated by
   bisection and stored in the vector X[1:N]. EPS1 is the accuracy
   required and RELFEH is the number closest to machine zero. The
   total number of iterations is stored in Z and EPS2 gives the actual
   accuracy obtained;
'VALUE' C,D,N,M1,M2,P,EPS1,RELFEH;
'ARRAY' C,B,D,X;
'REAL' EPS1,EPS2,RELFEH;
'INTEGER' N,P,M1,M2;
'BEGIN'
     'ARRAY' R[P:N,3:P],U,V[1:N];
     'REAL' XMIN,XMAX,H;
     'INTEGER' I,J;
     B[1]←0;
     H←ABS(B[N])+ABS(D[N]);
     XMAX←C[N]+H;
     XMIN←C[N]-H;
     'FØR' I←N-1 'STEP' -1 'UNTIL' 1 'DØ'
     'BEGIN'
          H←ABS(B[I])+ABS(B[I+1]);
          'IF' P-I+1 'LE' N 'THEN'
          H←H+ABS(D[P-1+I]);
          'IF' I 'GE' P 'THEN'
          H←H+ABS(D[I]);
          'IF' C[I]+H 'GT' XMAX 'THEN' XMAX←C[I]+H;
          'IF' C[I]-H 'LT' XMIN 'THEN' XMIN←C[I]-H;
     'END';
     EPS2←RELFEH*('IF' XMIN+XMAX 'GT' 0 'THEN' XMAX 'ELSE' -XMIN);
     'IF' EPS1 'LE' 0 'THEN' EPS1←EPS2;
     EPS2←0.5*EPS1+7*EPS2;
     'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
     R[I,P]←D[I];
     'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
     V[I]←B[I];
     'BEGIN'
          'ARRAY' WU[M1:M2]; 'REAL' S,X1,XU,X0;
          'INTEGER' T,A,K,L;
          X0←XMAX;
          'FØR' I←M1 'STEP' 1 'UNTIL' M2 'DØ'
          'BEGIN'
               X[I]←XMAX; WU[I]←XMIN;
          'END';
          Z←0;
          'FØR' L←M2 'STEP' -1 'UNTIL' M1 'DØ'
          'BEGIN'
               XU←XMIN;
               'FØR' I←L 'STEP' -1 'UNTIL' M1 'DØ'
```

```
                    'BEGIN'
                         'IF' XU 'LT' WU[I] 'THEN'
                         'BEGIN'
                              XU←WU[I]; 'GØTØ' CØNTIN;
                         'END';
                    'END';
CØNTIN:             'IF' XO 'GT' X[L] 'THEN' XO←X[L];
                    'FØR' X1←(XU+XO)/2 'WHILE' XO-XU 'GT'
                         2*RELFEH*(ABS(XU)+ABS(XO))+EPS1 'DØ'
                    'BEGIN' Z←Z+1;
                         U[1] 'IF' C[1]-X1 'EQ' 0 'THEN' RELFEH 'ELSE' C[1]-X1;
                         'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
                         'BEGIN'
                              U[I]←C[I]-X1-V[I]*V[I]/U[I-1];
                              'IF' U[I] 'EQ' 0 'THEN' U[I]-RELFEH;
                         'END';
                         'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ';
                         'BEGIN'
                              S←R[I,P]←R[I,P]*V[I-P+2]/U[I-P+1];
                              'FØR' K←P-2 'STEP' -1 'UNTIL' 3 'DØ'
                              'BEGIN'
                                   T←I-K+1;
                                   S←-S*V[T]/U[T-1];
                                   'IF' T 'GE' P 'THEN'
                                   'FØR' J←P-K-2 'STEP' -1 'UNTIL' 0 'DØ'
                                   S←S-R[I,P-J]*R[T,P+1-K-J]/U[I-P+J+1];
                                   R[I,K]←S;
                              'END';
                              S←B[I]-S*V[I-1]/U[I-2];
                              'IF' I-1 'GE' P 'THEN'
                              'FØR' J←P-4 'STEP' -1 'UNTIL' 0 'DØ'
                              S←S-R[I,P-J]*R[I-1,P-1-J]/U[I-P+J+1];
                              V[I]←S;
                              S←C[I]-X1-S*V[I]/U[I-1];
                              'FØR' J←P-3 'STEP' -1 'UNTIL' 0 'DØ'
                              S←S-R[I,P-J]*R[I,P-J]/U[I+P+J+1];
                              'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE'
                              U[I]←RELFEH;
                         'END'; A←0;
                         'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
                         'IF' U[I] 'LT' 0 'THEN' A←A+1;
                         'IF' A 'LT' L 'THEN'
                         'BEGIN'
                              'IF' A 'LT' M 'THEN' XU←
                                   WU[M1]←X1; 'ELSE'
                              'BEGIN'
                                   XU←WU[A+1]←X1;
                                   'IF' X[A] 'GT' X1 'THEN'
                                        X[A]←X1;
                              'END';
                         'END'
                         'ELSE' XO←X1;
                    'END';
                    X[L]←(XO+XU)/2;
               'END';
          'END';
'END';
```

PROGRAM 12A

This program determines the eigenvalues of a symmetric quin-diagonal matrix of semi-bandwidth P, using the modified bisection algorithm of Chapter 5. Also included are the modifications suggested by Tewarson for small element replacement.

```
'PRØCEDURE' MBQS(C,B,D,N,N1,N2,P,EPS,QZR,EIG);
'CØMMENT' C is the diagonal, B the sub-diagonal, and D the diagonal of
     semi-bandwidth P of the matrix of order N. The eigenvalues N1 to
     N2 (N2≥N1, eigenvalue 1 being the smallest) are calculated by the
     modified bisection method and stored in EIG[1:N2-N1+1]. EPS1 is
     the accuracy required and QZR the number closest to machine zero
     (used to prevent zero divisions);
'ARRAY' C,B,D,EIG;
'REAL' EPS,QZR;
'INTEGER' N,P,N1,N2;
'BEGIN'
     'ARRAY' R[P:N,3:P],U,V[1:N];
     'REAL' MIN,MAX,S,H,NEW,NEWER,LAMBDA;
     'INTEGER' I,J,T,K,L,M;
     B[1]←0;
     H←ABS(B[N])+ABS(D[N]);
     MAX←C[N]+H;
     MIN←C[N]-H;
     'FØR' I←N-1 'STEP' -1 'UNTIL' 1 'DØ'
     'BEGIN'
          H←ABS(B[I]+ABS(B[I+1]);
          'IF' P-I+1 'LE' N 'THEN'
          H←H+ABS(D[P-1+I]);
          'IF' I 'GE' P 'THEN'
          H←H+ABS(D[I]);
          'IF' C[I]+H 'GT' MAX 'THEN' MAX←C[I]+H;
          'IF' C[I]-H 'LT' MIN 'THEN' MIN←C[I]-H;
     'END';
     'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
     R[I,P]←D[I];
     'FØR' I←1 'STEP' 1 'UNTIL' P-1 'DØ'
     V[I]←B[I];
     L←N1-1;
LABA: 'IF' L 'GE' N2 'THEN' 'GØTØ' EXIT;
     LAMBDA←(MIN+MAX)*0.5;
LABB: M←0;
     U[1]←C[1]-LAMBDA;
     'IF' U[1] 'LT' 0 'THEN' M←M+1;
     'IF' L-M 'LT' 0 'OR' L-M-N+I 'GT' 0 'THEN' 'GØTØ' LABC;
     'IF' U[1] 'EQ' 0 'THEN' U[1]←QZR;
     'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
     'BEGIN'
          S←C[I]-LAMBDA-V[I]*V[I]/U[I-1];
          'IF' S 'LT' 0 'THEN' M←M+1;
          'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GT' 0 'THEN' 'GØTØ' LABC;
```

```
          'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
     'END';
     'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          S←-R[I,P]*V[I-P+2]/U[I-P+1];
          'IF' ABS(S) 'LT' 0.0000001 'THEN' S←R[I,P-1]←0 'ELSE' R[I,P]←S;
          'FØR' K←P-2 'STEP' -1 'UNTIL' 3 'DØ'
          'BEGIN'
               T←I-K+1;
               S←S*V[T]/U[T-1];
               'IF' T 'GE' P 'THEN'
               'FØR' J←P-K-2 'STEP' -1 'UNTIL' 0 'DØ'
               S←S-R[I,P-J]*R[T,P+1-K-J]/U[I-P+J+1];
               'IF' ABS(S) 'GT' 0.0000001 'THEN' R[I,K]←S 'ELSE' R[I,K]←S←0;
          'END';
          S←B[I]-S*V[I-1]/U[I-2];
          'IF' I-1 'GE' P 'THEN'
          'FØR' J←P-4 'STEP' -1 'UNTIL' 0 'DØ'
          S←S-R[I,P-J]*R[I-1,P-1-J]/U[I-P+J+1];
          'IF' ABS(S) 'GT' 0.0000001 'THEN' V[I]←S 'ELSE' V[I]←S←0;
          S←C[I]-LAMBDA-S*S/U[I-1];
          'FØR' J←P-3 'STEP' -1 'UNTIL' 0 'DØ'
          S←S-R[I,P-J]*R[I,P-J]/U[I-P+J+1];
          'IF' S 'LT' 0 'THEN' M←M+1;
          'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GT' 0 'THEN' 'GØTØ' LABC;
          'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
     'END';
LABC: 'IF' L-M 'GE' 0 'THEN'
     'BEGIN'
      → NEWER←MIN;
          MIN←LAMBDA;
          LAMBDA←LAMBDA+(LAMBDA-NEWER)*0.5;
     'END'
     'ELSE'
     'BEGIN'
          LAMBDA←(LAMBDA+MIN)*0.5
          'IF' (LAMBDA-MIN)*0.5/ABS(LAMBDA) 'LT' EPS
          'THEN'
          'BEGIN'
               MIN←2*LAMBDA-MIN;
               'FØR' I←L+1 'STEP' 1 'UNTIL'
               ('IF' M 'GT' N2 'THEN' N2 'ELSE' M)
               'DØ' EIG[I-N1+1]←LAMBDA;
               L←M;
               'GOTO' LABA;
          'END';
      'END';
      'GØTØ' LABB;
EXIT:
'END';
```

PROGRAM 13

The following program is the ALGOL 60 program to determine the eigenvectors of a banded symmetric quindiagonal matrix of semi-bandwidth P using inverse iteration.

```
'PRØCEDURE' EIGVEC (C,B,D,N,P,M,EIG,VEC);
'CØMMENT' C is the diagonal, B the sub-diagonal and D the diagonal of semi-
      bandwidth P of the (N×N) matrix N>4. There are M eigenvectors
      required with corresponding eigenvalues stored in EIG. When the
      eigenvectors have been determined by inverse iteration they are
      stored in the columns of VEC.
'VALUE' C,B,D,N,M,P,EIG;
'ARRAY' C,B,D,EIG,VEC;
'INTEGER' N,P,M;
'BEGIN'
      'ARRAY' V[1:N,P-1:N],U[P:N,1:P-2],W,Y,[2:P],X[1:P],Z[3:P],VT[1:N];
      'REAL' S;
      'INTEGER' 'ARRAY' IC[1:N-1];
      'INTEGER' I,J,K,L,R,T;
      X[P-1]←Y[P-1]←Z[P]←0;
      'FØR R←1 'STEP' 1 'UNTIL' M 'DØ'
      'BEGIN'
           'CØMMENT' This sets up the matrix in U,V,W,X,Y,Z and sets
                the initial vector VT to all 1's.
           'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
           VT[I]←1;
           T←0;
           'FØR' I←2 'STEP' 1 'UNTIL' P-2 'DØ'
           'BEGIN'
                X[I]←C[I]-EIG[R];
                Y[I]←W[I]←B[I];
                Z[I+1]←0;
           'END';
           W[P-1]←B[P-1];
           X[1]←C[1]-EIG[R];
           'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
           'BEGIN'
                'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
                V[I,J]←0;
           'END';
           'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
           'BEGIN'
                'FØR' J←1 'STEP' 1 'UNTIL' P-2 'DØ'
                U[I,J]←0;
           'END';
           'IF' P-3 'LT' N-P 'THEN'
           'BEGIN'
                'FØR' I←0 'STEP' 1 'UNTIL' P-3 'DØ'
                U[P+I,I+1]←D[P+I];
                'FØR' I←P-2 'STEP' 1 'UNTIL' N-P 'DØ'
                V[P+I,I+1]←D[P+I];
                'FØR' I←0 'STEP1 'UNTIL' N-P 'DØ'
```

```
     V[I+1,P+I]←D[P+I];
'END'
'ELSE'
'BEGIN'
     'FØR' I←O 'STEP' 1 'UNTIL' N-P 'DØ'
     U[P+I,I+1]←V[I+1 ,P+1]←D[P+I];
'END';
'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
'BEGIN'
     V[I,I]←C[I]-EIG[R];
     V[I,I-1]←V[I-1,I]←B[I];
'END';
V[P-1,P-1]←C[P-1]-EIG[R];
V[P-2,P-1]←B[P-1];
'FØR' I←1 'STEP' 1 'UNTIL' P-4 'DØ'
'BEGIN'
'CØMMENT' First stage of interchange and elimination of
     variables in rows 1-P-4.  Interchanges noted in IC;
     L←O; S←ABS(X[I]);
     'IF' S 'LT ABS(W[I+1]) 'THEN'
     'BEGIN'
          L←I+1; S←ABS(W[I+1]);
     'END';
     'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
     'IF' S 'LT' ABS(U[J,I]) 'THEN'
     'BEGIN'
          L←J; S←ABS(U[J,I]);
     'END';
     IC[I]←L;
     'IF' L 'EQ' I+1 'THEN'
     'BEGIN'
          S←X[I]; X[I]←W[L]; W[L]←S;
          S←Y[L]; Y[L]←X[L]; X[L]←S;
          S←Z[I+2]; Z[I+2]←Y[I+2]; Y[I+2]←S;
          'FØR' J←P-1 'STEP' 1 ' UNTIL' N 'DØ'
          'BEGIN'
               S←V[I,J]; V[I,J]←V[L,J];
               V[L,J]←S;
          'END'
          'ELSE'
          'IF' L 'NE' O 'THEN'
          'BEGIN'
               S←X[I]; X[I]←U[L,I];
               U[L,I]←S;
               S←Y[I+1]; Y[I+1]←U[L,I+1];
               U[L,I+1]←S;
               S←Z[I+2]; Z[I+2]←U[L,I+2];
               U[L,I+2]←S;
               'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
               'BEGIN'
                    S←V[I,J]; V[I,J]←V[L,J];
                    V[L,J]←S;
               'END';
          'END';
          'BEGIN'
               S←W[I+1]←W[I+1]/X[I];
```

```
                X[I+1]←X[I+1]-S*Y[I+1];
                Y[I+2]←Y[I+2]-S*Z[I+2];
                'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
                V[I+1,J]←V[I+1,J]-S*V[I,J];
                'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
                'BEGIN'
                     'IF' U[J,I] 'NE' 0 'THEN'
                     'BEGIN'
                          S←U[J,I]←U[J,I]/X[I];
                          U[J,I+1]←U[J,I+1]-S*Y[I+1];
                          U[J,I+2]←U[J,I+2]-S*Z[I+2];
                          'FØR' K←P-1 'STEP' 1 'UNTIL' N 'DØ'
                          V[J,K]←V[J,K]-S*V[I,K];
                     'END';
                'END';
           'END';
     'END';
'END';
'CØMMENT' Elimination of variables P-3 and P-2 performed
     seperately as diagonals W,X,Y,Z merge into V;
I←P-3; L←0; S←ABS(X[I]);
'IF' S 'LT' ABS(W[I+1]) 'THEN'
'BEGIN'
     L←I+1; S←ABS(W[I+1]);
'END';
'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
'IF' S 'LT' ABS(U[J,I]) 'THEN'
'BEGIN'
     L←J; S←ABS(U[J,I]);
'END';
IC[I]←L;
'IF' L 'EQ' I+1 'THEN'
'BEGIN'
     S←X[I]; X[I]←W[L]; W[L]←S;
     S←Y[L]; Y[L]←X[L]; X[L]←S;
     'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          S←V[I,J]; V[I,J]←V[L,J];
          V[L,J]←S;
     'END';
'END'
'ELSE'
'IF' L 'NE' 0 'THEN'
'BEGIN'
     S←X[I]; X[I]←U[L,I]; U[L,I]←S;
     S←Y[I+1]; Y[I+1]←U[L,I+1]; U[L,I+1]←S;
     'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          S←V[I,J];V[I,J]←V[L,J]; V[L,J]←S;
     'END';
'END'
'ELSE'
'IF' L 'NE' 0 'THEN'
'BEGIN'
     S←X[I]; X[I]←U[L,I]; U[L,I]←S;
     S←Y[I+1]; Y[I+1]←U[L,I+1]; U[L,I+1]←S;
     'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
```

```
     'BEGIN'
          S←V[I,J]; V[I,J]←V[L,J]; V[L,J]←S;
     'END';
'END';
S←W[I+1]←W[I+1]/X[I];
X[I+1]←X[I+1]-S*Y[I+1];
'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
V[I+1,J]←V[I+1,J]-S*V[I,J];
'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ;
'BEGIN'
     'IF' U[J,I] 'NE' 0 'THEN'
     'BEGIN'
          S←U[J,I]←U[J,I]/X[I];
          U[J,I+1]←U[J,I+1]-S*Y[I+1];
          'FØR' K←P-1 'STEP' 1 'UNTIL' N 'DØ'
          V[J,K]←V[J,K]-S*V[I,K];
     'END';
'END';
I←P-2; L←0; S←ABS(X[I]);
'IF' S 'LT' ABS(W[I+1]) 'THEN'
'BEGIN'
     L←I+1; S←ABS(W(I+1]);
'END';
'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
'IF' S 'LT' ABS(U[J,I]) 'THEN'
'BEGIN'
     L←J; S←ABS(U[J,I]);
'END';
IC[I]←L;
'IF' L 'EQ' I+1 'THEN'
'BEGIN'
     S←X[I]; X[I]←W[L]; W[L]←S;
     'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          S←V[I,J]; V[I,J]←V[L,J]; V[L,J]←S;
     'END';
'END'
'ELSE'
'IF' L 'NE' 0 'THEN'
'BEGIN'
     S←X[I];X[I]←U[L,I];U[L,I]←S;
     'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
     'BEGIN'
          S←V[I,J]; V[I,J]←V[L,J]; V[L,J]←S;
     'END';
'END';
S←W[I+1]←W[I+1]/X[I];
'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
V[I+1,J]←V[I+1,J]-S*V[I,J];
'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
'BEGIN'
     'IF' U[J,I] 'NE' 0 'THEN'
     'BEGIN'
          S←U[J,I]←U[J,I]/X[I];
          'FØR' K←P-1 'STEP' 1 'UNTIL' N 'DØ'
          V[J,K]←V[J,K]-S*V[I,K];
     'END';
'END';
```

```
          'CØMMENT' The variables to be eliminated are now contained
               in the lower P-1 rows of V, and normal Gaussian
               elimination is now performed;
          'FØR' I←P-1 'STEP' 1 ' UNTIL' N-1 'DØ'
          'BEGIN'
               L←O; S←ABS(V[I,I]);
               'FØR' J←I 'STEP' 1 'UNTIL' N 'DØ'
               'IF' S 'LT' ABS(V[J,I]) 'THEN'
               'BEGIN'
                    L←J; S←ABS(V[J,I]);
               'END';
               IC[I]←L;
               'IF' L 'NE' O 'THEN'
               'FØR' J←I 'STEP' 1 'UNTIL' N 'DØ'
               'BEGIN'
                    S←V[I,J]; V[I,J]←V[L,J]; V[L,J]←S;
               'END';
               'FOR' J←I+1 'STEP' 1 'UNTIL' N 'DO'
               'IF' V[J,I] 'NE' O 'THEN'
               'BEGIN'
                    S←V[J,I]←V[J,I]/V[I,I];
                    'FØR' L←I+1 'STEP' 1 'UNTIL' N 'DØ'
                    V[J,L]←V[J,L]-S*V[I,L];
               'END';
          'END';
LAB1:     'IF' V[N,N] 'EQ' O 'THEN' V[N,N]←RELFEH;
          'CØMMENT' to avoid division by zero in the decomposing case
               the last element is replaced by a small number if it is
               zero.  Back substitution now takes place;
          VT[N]←VT[N]/V[N,N];
          'FØR' I←N-1 'STEP' -1 'UNTIL' P-1 'DØ'
          'BEGIN'
               S←VT[I];
               'FØR' J←I+1 'STEP' 1 'UNTIL' N 'DØ'
               S←S-V[I,J]*VT[J];
               VT[I]←S/V[I,I];
          'END';
          'FØR' I←P-2 'STEP' -1 'UNTIL' 1 'DØ'
          'BEGIN'
               S←VT[I];
               'FØR' J←P-1 'STEP' 1 'UNTIL' N 'DØ'
               S←S-V[I,J]*VT[J];
               VT[I]←(S-Y[I+1]*VT[I+1]-Z[I+2]*VT[I+2])/X[I];
          'END';
          S←O;
          FO1ARA(1,1,N,O,O,S,VT[I],VT[I],I,S,S);
          'CØMMENT' A N.A.G. subroutine is used here to perform double
               length accumulation of products whilst normalising the
               vector.
          S←1/SQRT(S);
          'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
          VT[I]←VT[I]*S;
          'IF' T 'EQ' 1 'THEN' 'GØTØ' LAB2;
          'CØMMENT' Elimination with interchanges now takes place using
               stored information in two major steps;
          'FØR' I←1 'STEP' 1 'UNTIL' P-2 'DØ'
          'BEGIN'
```

```
                'IF' IC[I] 'NE' 0 'THEN'
                'BEGIN'
                    S←VT[I]; VT[I]←VT[IC[I]];
                    VT[IC[I]]←S;
                'END';
                S←VT[I];
                VT[I+1]←VT[I+1]-S*W[I+1];
                'FØR' J←P 'STEP' 1 'UNTIL' N 'DØ'
                VT[J]←VT[J]-S*U[J,I];
            'END';
            'FØR' I←P-1 'STEP' 1 'UNTIL' N-1 'DØ'
            'BEGIN'
                'IF' IC[I] 'NE' 0 'THEN'
                'BEGIN'
                    S←VT[I]; VT[I]←VT[IC[I]];
                    VT[IC[I]]←S;
                'END';
                S←VT[I];
                'FØR' J←I+1 'STEP' 1 'UNTIL' N 'DØ'
                VT[J]←VT[J]-S*V[J,I];
            'END';
            T←T+1;
            'GØTØ' LAB1;
LAB2:       'FØR' J←1 'STEP' 1 'UNTIL' N 'DØ'
            VEC[J,R]←VT[J];
        'END';
'END';
```

PROGRAM 14

This program finds the eigenvalues of an unsymmetric banded quin-diagonal matrix of semi-bandwidth P using Mullers method. Also included are the two routines used to perform a complex division, and to find the square root of a complex number.

```
'PRØCEDURE' MULLER(C,B,D,E,F,N,P,EPS,QZR,EIG,EIGI);
'CØMMENT' The main diagonal of the matrix of order N is stored in the
      vector C. The two sub-diagonals are stored in B and D, and the
      bands at the semi-bandwidth P are stored in E and F. The accuracy
      required is placed in EPS and QZR is the smallest number above
      machine zero to replace zero divisors. The eigenvalues are
      placed in EIG,EIGI when determined;
'VALUE' C,B,D,E,F,N,P,EPS,QZR;
'ARRAY' C,B,D,E,F,EIG,EIGI;
'REAL' EPS,QZR;
'INTEGER' N,P;
'BEGIN'
      'ARRAY R,RI,RL,RLI[P:N,3:P],U,UI,V,VI,VL,VLI,SN,SNI,TN,TNI[1:N];
      'INTEGER' T,B1,I,J,K,L,M,Z;
      'REAL' X1,X2,X3,F1,F2,F3,X1I,X2I,X3I,F1I,F2I,F3I,X,XI,G,GI,A,AI,W,
          WI,Y,YI,VS,VSI,INT,INTI,H,HI,S,SI,SL,SLI;
      'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
      'CØMMENT' The vectors for calculating and storing the Sturm
          sequence are initialised;
      'BEGIN'
          R[I,P]←D[I];
          RL[I,P]←F[I];
          RI[I,P]←RLI[I,P]←0;
      'END';
      'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
      'BEGIN'
          VI[I]←VLI[I] 0;
          VL[I]←E[I];
          V[I]←B[I];
      'END';
      B1←0;
      V[1]←VI[1]←VL[1]←VLI[1]←0;
      L←1;
      Z←0;
      X3←C[1]+0.1;
      X3I←0;
LAB1: 'BEGIN'
          Z←Z+1
          'IF' C[1]-X3 'EQ' 0 'AND' X3I 'EQ' 0 'THEN'
          'BEGIN'
              U[1]←UI[1]←QZR;
          'END'
          'ELSE'
          'BEGIN'
              U[1]←C[1]-X3; UI[1]←-X3I;
          'END';
          'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
```

```
'BEGIN'
     INT←V[I]*VL[I]-VI[I]*VLI[I];
     INTI←V[I]*VLI[I]+VL[I]*VI[I];
     DIV(INT,INTI,U[I-1],UI[I-1],INT,INTI);
     U[I]←C[I]-X3-INT;
     UI[I]←-X3I-INTI;
     'IF' U[I] 'EQ' 0 'AND' UI[I] 'EQ' 0 'THEN'
     U[I]←QZR;
'END';
'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
'BEGIN'
     DIV[VL[I-P+2],VLI[I-P+2],U[I-P+1],UI[I-P+1],
     INT,INTI);
     DIV(V[I-P+2],VI[I-P+2],U[I-P+1],UI[I-P+1],
     H,HI);
     R[I,P-1]←-R[I,P]*INT+RI*INTI;
     RI[I,P-1]←-RI[I,P]*INT-R[I,P]*INTI;
     RL[I,P-1]←-RL[I,P]*H+RLI[I,P]*HI;
     RLI[I,P-1]←-RLI[I,P]*H-RL[I,P]*HI;
     'FØR' K←P-2 'STEP' -1 'UNTIL' 3 'DØ'
     'BEGIN'
          T←I-K+1;
          DIV(VL[T],VLI[T],U[T-1],UI[T-1],INT,INTI);
          DIV(V[T],VI[T],U[I-1],UI[T-1],H,HI);
          S←-R[I,K+1]*INT+RI[I,K+1]*INTI;
          SI←-R[I,K+1]*INTI-RI[I,K+1]*INT;
          SL←-RL[I,K+1]*H+RLI[I,K+1]*HI;
          SLI←-RL[I,K+1]*HI-RLI[I,K+1]*H;
          'IF' T 'GE' P 'THEN'
          'FØR' J←P-K-2 'STEP' -1 'UNTIL' 0 'DØ'
          'BEGIN'
               DIV(RL[T,P+1-K-J],RLI[T,P+1-K-J],
               U[I-P+J+1],UI[I-P+J+1],INT,INTI);
               DIV(R[T,P+1-K-J],RI[T,P+1-K-J],
               U[I-P+J+1],UI[I-P+J+1],H,HI);
               S←S-R[I,P-J]*INT+RI[I,P-J]*INTI;
               SI←SI-R[I,P-J]*INTI-RI[I,P-J]*INT;
               SL←SL-RL[I,P-J]*H+RLI[I,P-J]*HI;
               SLI←SLI-RLI[I,P-J]*H-RL[I,P-J]*HI;
          'END';
          R[I,K]←S;
          RI[I,K]←SI;
          RL[I,K]←SL;
          RLI[I,K]←SLI;
     'END';
     DIV(VL[I-1],VLI[I-1],U[I-2],UI[I-2],INT,INTI);
     DIV(V[I-1],VI[I-1],U[I-2],UI[I-2],H,HI);
     S←B[I]-R[I,3]*INT+RI[I,3]*INTI;
     SI←-R[I,3]*INTI-RI[I,3]*INT;
     SL←E[I]-RL[I,3]*H+RLI[I,3]*HI;
     SLI←-RL[I,3]*HI-RLI[I,3]*H;
     'IF' I-1 'GE' P 'THEN'
     'FØR' J←P-4 'STEP' -1 'UNTIL' 0 'DØ'
     'BEGIN'
          DIV(RL[I-1,P-1-J],RLI[I-1,P-1,J],
          U[I-P+J+1],UI[I-P+J+1],INT,INTI);
          DIV(R[I-1,P-1-J],RI[I-1,P-1-J],
          U[I-P+J+1],UI[I-P+J+1],H,HI);
```

```
            S←S-R[I,P-J];*INT+RI[I,P-J]*INTI;
            SI←SI-R[I,P-J]*INTI-RI[I,P-J]*INTI;
            SL←SL-RL[I,P-J]*H+RLI[I,P-J]*HI;
            SLI←SLI-RLI[I,P-J]*HI-RL[I,P-J]*H ;
        'END';
        V[I]←S;
        VI[I]←SI;
        VL[I]←SL;
        VLI[I]←SLI;
        DIV(V[I],VI[I],U[I-1],UI[I-1],INT,INTI);
        S←C[I]-X3-VL[I]*INT+VLI[I]*INTI;
        SI←X3I-VL[I]*INTI-VLI[I]*INT;
        'FØR' J←P-3 'STEP' -1 'UNTIL' 0 'DØ'
        'BEGIN'
            DIV(R[I,P-J],RI[I,P-J],U[I-P+J+1],
            UI[I-P+J+1],H,HI);
            S←S-RL[I,P-J]*H+RLI[I,P-J]*HI;
            SI←SI-RL[I,P-J]*HI-RLI[I,P-J]*H;
        'END';
        U[I]←S;
        UI[I]←S;
        'IF' U[I] 'EQ' 0 'AND' UI[I] 'EQ' 0 'THEN' U[I]←QZR;
'END';
'CØMMENT' After finding the Sturm sequence it is "deflated"
    to prevent redetermination of known eigenvalues;
'FØR' J←2 'STEP' 1 'UNTIL' L 'DØ'
'BEGIN'
    H←EIG[J-1]-X3;
    HI←EIGI[J-1]-X3I;
    DIV(U[N+2-J],UI[N+2-J],H,HI,U[N+2-J],UI[N+2-J]);
'END';
F3←U[N];
F3I←UI[N];
'FØR' J←N-1 'STEP' -1 'UNTIL' N-L+1 'DØ'
'BEGIN'
    H←F3; HI←F3I;
    F3←H*U[J]-HI*UI[J];
    F3I←HI*U[J]+H*UI[J];
'END';
'CØMMENT' As three function evaluations at three different points
    are required for Mullers method at each step, two are chosen
    randomly.  The third is found from the previous two using the
    Secant formula, and the method can then proceed normally.
    The following code governs this process.
'IF' B1 'LT' 2 'THEN'
'BEGIN'
    'IF' B1 'EQ' 0 'THEN'
    'BEGIN'
        B1←1;
        'FØR' J←1 'STEP' 1 'UNTIL' N 'DØ'
        'BEGIN'
            SN[J]←U[J]; SNI[J]←UI[J];
        'END';
        XL←X3; XII←X3I; X3←C[1]+1;
        F1←U[N]; F1I←UI[N];
        'GØTØ' LAB1;
```

```
     'END'
     'ELSE'
     'BEGIN'
          B1←2;
          'FØR' J←1 'STEP' 1 'UNTIL' N 'DØ'
          'BEGIN'
               TN[J]←U[J];
               TNI[J]←UI[J];
          'END';
          X2←X3; X2I←X3I;
          F2←U[N]; F2I←UI[N];
          H←TN[N]-SN[N];
          HI←TNI[N]-SNI[N];
          INT←X2-X1;
          INTI←X2I-X1I;
          DIV(H,HI,INT,INTI,W,WI);
          DIV(TN[N],TNI[N],W,WI,H,HI);
          X3←X2-H;
          X3I←X2I-HI;
          X←X3-X2;
          XI←X3I-X2I;
          A←X2-X1;
          AI←X2I-X1I;
          DIV(X,XI,A,AI,A,AI);
          'GØTØ' LAB1;
     'END';
'END';
'CØMMENT' Mullers formula is now calculated using three previous
     function evaluations at points within the eigenvalue spectrum;
G←(1+2*A)*(F3-F2)-(2*AI)*(F3I-F2I)-(A*A-AI*AI)*
     (F2-F1)+2*A*AI*(F2I-F1I);
GI←(1+2*A)*(F3I-F2I)+2*AI*(F3-F2)-(A*A-AI*AI)*
     (F2I-F1I)-(F2-F1)*2*A*AI;
W←A*(F2-F1)-AI*(F2I-F1I);
WI←AI*(F2-F1)+A*(F2I-F1I);
Y←(F3-F2-W)*A-(F3-F2I-WI)*AI;
YI←(F3-F2-W)*AI+(F3I-F2I-WI)*A;
VS←(1+A)*Y-AI*YI;
VSI←(1+A)*YI+AI*Y;
W←4*(VS*F3-VSI*F3I);
WI←4*(VSI*F3+VS*F3I);
Y←G*G-GI*GI-W;
YI←2*G*GI-WI;
ISQRT(Y,YI,Y,YI);
H←G-Y;
HI←GI-YI;
INT←G+Y;
INTI←GI+YI;
W←H*H+HI*HI;
WI←INT*INT+INTI*INTI;
VS←-2*F3*(1+A)+2*F3I*AI;
VSI←-2*(F3*AI+F3I*(1+A));
'IF' W 'GT' WI 'THEN'
DIV(VS,VSI,H,HI,A,AI)
'ELSE'
DIV(VS,VSI,INT,INTI,A,AI);
W←X*A-XI*AI;
WI←XI*A+X+AI;
H←SQRT(W*W+WI*WI);
'IF' H/SQRT(X3*X3+X3I*X3I) 'LT' EPS 'THEN'
```

```
          'BEGIN'
               EIG[L]←X3+W;
               EIGI[L]←X3I+WI;
               INT←EIG[L]-X1;
               INTI←EIGI[L]-X11;
               DIV(F1,F1I,INT,INTI,F1,F11);
               INT←EIG[L]-X2;
               INTI←EIGI[L]-X21;
               DIV(F2,F2I,INT,INTI,F2,F2I);
               L←L+1;
               Z←O;
               H←F2-F1;
               HI←F2I-F1I;
               INT←X2-X1;
               INTI←X2I-X1I;
               'IF' INT 'EQ' O 'AND' INTI 'EQ' O
               'THEN' INT←EPS;
               X←W;
               XI←WI;
               DIV(H,HI,INT,INTI,H,HI);
               DIV(F2,F2I,H,HI,H,HI);
               X3←X2-H;
               X31←X2I-HI;
               B1←1;
               'IF' L 'EQ' N+1 'THEN' 'GØTØ' LAB2;
               'GØTØ' LAB1;
          'END'
          'ELSE'
          'BEGIN'
               X1←X2; X1I←X2I;
               X2←X3; X2I←X3I;
               X3←X2+W; X3I←X2I+W;
               X←W; XI←WI;
               F1←F2; F1I←F2I;
               F2←F3; F2I←F3I;
               'FØR' J←1 'STEP' 1 'UNTIL' N 'DØ'
               'BEGIN'
                    SN[J]←TN[J]; SNI[J]←TNI[J];
                    TN[J]←U[J]; TNI[J]←UI[J];
               'END';
               'GØTØ' LAB1;
          'END';
LAB2:'END';
'END';


'PROCEDURE' DIV(A,B,C,D,E,F);
'VALUE' A,B,C,D;
'REAL' A,B,C,D,E,F;
'BEGIN'
     'REAL' H;
     H←C*C+D*D;
     E←(A*C+B*D)/H;
     F←(C*B-A*D)/H;
'END';
```

```
'PROCEDURE' ISQRT(A,B,C,D);
'VALUE' A,B;
'REAL' A,B,C,D;
'BEGIN'
     'REAL' R,THI,P;
     R←SQRT(SQRT(A*A+B*B));
     PI←3.1415926536;
     'IF' A 'EQ' 0 'THEN'
     'BEGIN'
          'IF' B 'GT' 0 'THEN'
          THI←PI/4
          'ELSE'
          THI←3*PI/4;
     'END'
     'ELSE'
     'IF' B 'EQ' 0 'THEN'
     'BEGIN'
          'IF' A 'GT' 0 'THEN'
          THI←0
          'ELSE'
          THI←PI/2;
     'END'
     'ELSE'
     'IF' A 'GT' 0 'AND' B 'GT' 0 'THEN'
     THI←ARCTAN(B/A)/2
     'ELSE'
     'IF' A 'LT' 0 'AND' B 'GT' 0 'THEN'
     THI←(ARCTAN(-A/B)+PI/2)/2
     'ELSE'
     'IF' A 'GT' 0 'AND' B 'LT' 0 'THEN'
     THI←(ARCTAN(-A/B)+PI*1.5)/2
     'ELSE'
     THI←(ARCTAN(B/A)+PI)/2;
     C←R*COS(THI);
     D←R*SIN(THI);
'END';
```

PROGRAM 15

This procedure determines the eigenvalues of a periodic quindiagonal matrix using the bisection method. The Sturm sequence used is the space saving algorithm of section 6.2 in a modification of the bisection algorithm of program 1.

```
'PRØCEDURE' PQUINS(C,B,D,E,N1,N2,M,EPS,EPS1,QZR,EIG);
'CØMMENT' C is the main diagonal, B the sub-diagonal, and E the sub-
    sub-diagonal of the matrix of order N. The three corner elements
    are contained in the vector D. The eigenvalues N1 to N2 (N1≤N2,
    eigenvalue 1 being the smallest) are calculated by bisection and
    stored in the vector EIG[1:N]. EPS1 is the accuracy required,
    and QZR is the number closest to machine zero. The total number
    of iterations is stored M and EPS2 gives the actual accuracy
    attained;
'VALUE' C,B,D,E,N,N1,N2,EPS,QZR;
'ARRAY' C,B,D,E,EIG;
'REAL' EPS,EPS1,QZR,EIG;
'INTEGER' N,N1,N2,M;
'BEGIN'
    'REAL' MAX,MIN,NEW,NEWER,UNI,UN,VN,U1,V1,U2,V2,S,F,G,H,R,Q;
    'INTEGER' I,J,K,L,T;
    B[1]←E[1]←E[2]←0;
    'CØMMENT' The limits on the eigenvalue are now determined (MAX,MIN);
    'BEGIN'
        NEW←ABS(B[N])+ABS(E[N])+ABS(D[2])+ABS(D[3]);
        MAX←C[N]+NEW;
        MIN←C[N]-NEW;
        NEW←ABS(B[N])+ABS(B[N-1])+ABS(D[1])+ABS(E[N-1]);
        NEWER←C[N-1]+NEW;
        NEW←C[N-1]-NEW;
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        NEW←ABS(B[2])+ABS(D[1])+ABS(D[2])+ABS(E[3]);
        NEWER←C[1]+NEW;
        NEW←C[1]-NEW;
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        NEW←ABS(B[2])+ABS(B[3])+ABS(D[3])+ABS(E[4]);
        NEWER←C[2]+NEW;
        NEW←C[2]-NEW;
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'FØR' I←3 'STEP' 1 'UNTIL' N-2 'DØ'
        'BEGIN'
            NEW←ABS(B[I])+ABS(B[I+1])+ABS(E[I])+ABS(E[I+2]);
            NEWER←C[I]+NEW;
            NEW←C[I]-NEW;
            'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
            'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'END';
    'END';
```

```
     EPS1←QZR*('IF' MIN+MAX 'GT' 0 'THEN' XMAX 'ELSE'-XMIN);
     'IF' EPS 'LE' 0 'THEN' EPS←EPS1;
     EPS1←0.5*EPS+7*EPS1;
     V2←B[2];
     'BEGIN'
          'ARRAY' WU[N1:N2];'REAL' S,X1,XU,XO;
          'INTEGER' A;
          XO←MAX;
          'FØR' I←N1 'STEP' 1 'UNTIL' N2 'DØ'
          'BEGIN'
               X[I]←MAX; WU[I]←MIN;
          'END';
          M←0;
          'FØR' L←N2 'STEP' -1 'UNTIL' N1 'DØ'
          'BEGIN'
               XU←MIN;
               'FØR' I←L 'STEP' -1 'UNTIL' N1 'DØ'
               'BEGIN'
                    'IF' XU 'LT' WU[I] 'THEN'
                    'BEGIN'
                         XU←WU[I]; 'GØTØ' CØNTIN;
                    'END';
               'END';
CONTIN:        'IF' XO 'GT' X[L] 'THEN' XO←X[L];
               'FØR' X1←(XU+XO)/2 'WHILE' XO-XU 'GT'
                    2*QZR*(ABS(XU)+ABS(XO))+EPS 'DØ'
               'BEGIN'
                    M←M+1; A← 0;
                    'CØMMENT' calculation of the Sturm sequence now
                    takes place to determine the number of negative
                    elements in the sequence (A);
                    U1←'IF' C[1]-X1 'EQ' 0 'THEN' QZR 'ELSE' C[1]-X1;
                    'IF' U1 'LT' 0 'THEN A←A+1;
                    U2←C[2]-X1-V2*V2/U1;
                    'IF' U2 'LT' 0 'THEN' A←A+1;
                    'IF' U2 'EQ' 0 'THEN' U2←QZR;
                    G←-D[1]*B[2]/U1;
                    H←D[3]-B[2]*D[2]/U1;
                    R←-E[3]*D[1]/U1;
                    Q←-E[3]*D[2]/U1;
                    UN1←C[N-1]-X1-D[1]*D[1]/U1-G*G/U2;
                    UN←C[N]-X1-D[2]*D[2]/U1-H*H/U2;
                    VN←B[N]-D[2]*D[1]/U1-G*H/U2;
                    'FØR' I←3 'STEP' 1 'UNTIL' N-4 'DØ'
                    'BEGIN'
                         V2←B[I]-V2*E[I]/U1;
                         S←U2;
                         U2←C[I]-X1-V2*V2/S-E[I]*E[I]/U1;
                         U1←S;
                         'IF' U2 'LT' 0 'THEN' A←A+1;
                         'IF' U2 'EQ' 0 'THEN' U2←QZR;
                         F←V2/U1;
                         NEW←R-G*F;
                         NEWER←Q-H*F;
                         F←E[I+1]/U1;
                         R←-G*F;
                         Q←-H*F;
                         G←NEW;
                         H←NEWER;
```

```
                    UN1←UN1-G*G/U2;
                    UN←UN-H*H/U2;
                    VN←VN-G*H/U2;
                'END';
                V2←B[N-3]-V2*E[N-3]/U1;
                S←U2;
                U2←C[N-3]-X1-V2*V2/S-E[N-3]*E[N-3]/U1;
                U1←S;
                'IF' U2 'EQ' 0 'THEN' U2←QZR;
                'IF' U2 'LT' 0 'THEN' A←A+1;
                V1←B[N-2]-V2*E[N-2]/U1;
                S←U2;
                U2←C[N-2]-X1-V1*V1/S-E[N-2]*E[N-2]/U1;
                'IF' U2 'EQ' 0 'THEN' U2←QZR;
                'IF' U2 'LT' 0 'THEN' A←A+1;
                R←E[N-1]+R;
                F←V2/U1;
                R←R-G*F;
                Q←Q-H*F;
                F←E[N-2]/U1;
                V2←B[N-1]-G*F;
                H←E[N]-H*F;
                UN1←UN1-R*R/S;
                UN←UN-Q*Q/S;
                VN←VN-R*Q/S;
                F←V1/S;
                V2←V2-R*F;
                H←H-Q*F;
                UN1←UN1-V2*V2/U2;
                'IF' UN1 'EQ' 0 'THEN' UN1←QZR
                'IF' UN1 'LT' 0 'THEN' A←A+1;
                UN←UN-H*H/U2;
                VN←VN-H*V2/U2;
                UN←UN-VN*VN/UN1;
                'IF' UN 'LT' 0 'THEN' A←A+1;
                'IF' A 'LT' L 'THEN'
                'BEGIN'
                    'IF' A 'LT' N1 'THEN' XU←WU[N1]←X1
                    'ELSE'
                    'BEGIN'
                        XU←WU[A+1]←X1;
                        'IF' EIG[A] 'GT' X1 'THEN' EIG[A]←X1
                    'END';
                'END'
                'ELSE' XO←X1;
            'END';
            EIG[L]←(XO+XU)/2;
        'END';
    'END';
'END';
```

PROGRAM 16

This procedure determines the eigenvalues of a periodic quindiagonal matrix using the bisection method. The Sturm sequence used is the time saving algorithm of section 6.2 in a modified bisection algorithm as described in Chapter 5.

```
'PRØCEDURE' PQUINT(C,B,D,E,N,N1,N2,M,P,EPS,QZR,EIG);
'ARRAY' C,B,D,E,EIG;
'INTEGER' N,P,N1,N2,M;
'REAL' EPS,QZR;
'BEGIN'
    'ARRAY' U,V[1:N];
    'REAL' MAX,MIN,NEW,NEWER,LAMBDA,S,F,G,H,R,Q,U1,V1,UN,VN;
    'INTEGER' I,K,L,T;
    B[1]←E[1]←E[2]←0;
    'CØMMENT' The limits on the eigenvalues are now determined (MAX,MIN);
    'BEGIN'
        NEW←ABS(B[N])+ABS(E[N])+ABS(D[2])+ABS(D[3]);
        MAX←C[N]+NEW;
        MIN←C[N]-NEW;
        NEW←ABS(B[N])+ABS(B[N-1])+ABS(D[1])+ABS(E[N-1]);
        NEWER←C[N-1]+NEW;
        NEW←C[N-1]-NEW
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        NEW←ABS(B[2])+ABS(D[1])+ABS(D[2])+ABS(E[3]);
        NEWER←C[1]+NEW;
        NEW←C[1]-NEW;
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        NEW←ABS(B[2])+ABS(B[3])+ABS(D[3])+ABS(E[4]);
        NEWER←C[2]+NEW;
        NEW←C[2]-NEW;
        'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
        'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'FØR' I←3 'STEP' 1 'UNTIL' N-2 'DØ'
        'BEGIN'
            NEW←ABS(B[I])+ABS(B[I+1])+ABS(E[I])+ABS(E[I+2]);
            NEWER←C[I]+NEW;
            NEW←C[I]-NEW;
            'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
            'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'END';
    'END';
    V[2]←B[2];
    L←N1-1;
    'CØMMENT' Now the major loop is performed to obtain the L^th eigenvalue;
LABA:'IF' L 'GE' N2 'THEN' 'GØTØ' EXIT;
    LAMBDA←(MIN+MAX)*0.5;
LABB:'BEGIN' K←0;
        U[1]←'IF' C[1]-LAMBDA 'EQ' 0 'THEN' QZR 'ELSE' C[1]-LAMBDA;
        'IF' U[1] 'LT' 0 'THEN' K←K+1;
        'IF' L-K 'LT' 0 'ØR' L-K-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
```

```
          U[2]←C[2]-LAMBDA-V[2]*V[2]/U[1];
          'IF' U[2] 'EQ' 0 'THEN' U[2]←QZR;
          'IF' U[2] 'LT' 0 'THEN' K←K+1;
          'IF' L-K 'LT' 0 'ØR' L-K-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
          'FØR' I←3 'STEP' 1 'UNTIL' N-2 'DØ'
          'BEGIN'
               V[I]←B[I]-V[I-1]*E[I]/U[I-2];
               U[I]←C[I]-LAMBDA-V[I]*V[I]/U[I-1]-E[I]*E[I]/U[I-2];
               'IF' U[I] 'EQ' 0 'THEN' U[I]←QZR;
               'IF' 'LT' 0 'THEN' K←K+1;
               'IF' L-K 'LT' 0 'OR' L-K-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
          'END';
          G←-D[1]*B[2]/U[1];
          H←D[3]-B[2]*D[2]/U[1];
          R←-E[3]*D[1]/U[1];
          Q←-E[3]*D[2]/U[1];
          U1←C[N-1]-LAMBDA-D[1]*D[1]/U[1]-G*G/U[2];
          UN←C[N]-LAMBDA-D[2]*D[2]/U[1]-H*H/U[2];
          VN←B[N]-D[2]*D[1]/U[1]-G*H/U[2];
          'FØR' I←3 'STEP' 1 'UNTIL' N-4 'DØ'
          'BEGIN'
               F←V[I]/U[I-1];
               NEW←R-G*F;
               NEWER←Q-H*F;
               F←E[I+1]/U(I-1];
               R←-G*F;
               Q←-H*F;
               G←NEW;
               H←NEWER;
               U1←U1-G*G/U[I];
               UN←UN-H*H/U[I];
               VN←VN-G*H/U[I];
          'END';
          R←E[N-1]+R;
          F←V[N-3]/U[N-4];
          R←R-G*F;
          Q←Q-H*F;
          F←E[N-2]/U[N-4];
          V1←B[N-1]-G*F;
          H←E[N]-H*F;
          U1←U1-R*R/U[N-3];
          UN←UN-Q*Q/U[N-3];
          VN←VN-R*Q/U[N-3];
          F←V[N-2]/U[N-3];
          V1←V1-R*F;
          H←H-Q*F;
          U1←U1-V1*V1/U[N-2];
          'IF' U1 'EQ' 0 'THEN' U1←QZR;
          'IF' U1 'LT' 0 'THEN' K←K+1;
          UN←UN-H*H/U[N-2];
          VN←VN-H*V1/U[N-2];
          UN←UN-VN*VN/U1;
          'IF' UN 'LT' 0 'THEN' K←K+1;
     'END';
LABC:'IF' L-K 'GE' 0 'THEN'
     'BEGIN'
          NEWER←MIN;
          MIN←LAMBDA;
          LAMBDA←LAMBDA+(LAMBDA-NEWER)*0.5;
```

```
    'END'
    'ELSE'
    'BEGIN'
        LAMBDA←(LAMBDA+MIN)*0.5;
        'IF' (LAMBDA-MIN)*0.5/ABS(LAMBDA) 'LT' EPS 'THEN'
        'BEGIN'
            MIN←2*LAMBDA-MIN;
            'FØR' I←L+1 'STEP' 1 'UNTIL' ('IF' K 'GT' N2 'THEN'
                N2 'ELSE' K) 'DØ'
            EIG[I-N1+1]←LAMBDA;
            L←K;
            'GØTØ' LABA;
        'END';
    'END';
    'GØTØ' LABB;
EXIT:
'END';
```

PROGRAM 17

This procedure determines the eigenvectors of a symmetric periodic quindiagonal matrix using inverse iteration. The procedure is modified to economise on storage by taking advantage of the large number of zero elements.

```
'PRØCEDURE' PQVEC(C,B,E,A1,A2,A3,EIG,L,N,VEC);
'CØMMENT' C is the diagonal, B the sub-diagonal, and E the sub-sub-
     diagonal of the (N×N) input matrix. The elements in the corners
     corresponding to positions (1,N-1), (1,N), (2,N) are A1,A2,A3.
     EIG is where the L eigenvalues are input, and later overwritten
     by the Rayleigh quotient. The eigenvectors are stored in the
     columns of VEC.
'VALUE' C,B,E,A1,A2,A3,L,N;
'ARRAY' C,B,E,EIG,VEC;
'REAL' A1,A2,A3;
'INTEGER' L,N;
'BEGIN'
     'INTEGER' 'ARRAY' IC[1:N];
     'ARRAY' R,S,T,U,V,W,X,Y,Z,Q1,Q2,Q3,Q[1:N];
     'REAL' D,D1,D2,D3,D4;
     'INTEGER' I,J,K,M;
     'FØR' M←1 'STEP' 1 ' UNTIL' L 'DØ'
     'BEGIN'
          'CØMMENT' This is the major loop to determine the M^th
               eigenvector. First the vectors representing the input
               matrix are initialised;
          'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
          'BEGIN'
               T[I]←W[I]←X[I]←U[I]←0;
               V[I]←C[I]-EIG[M];
               IC[I]←0;
               Q[I]←1;
               Y[I]←Z[I]←Q1[I]←Q2[I]←Q3[I]←S[I]←R[I]←0;
          'END';
          'FØR' I←2 'STEP' 1 'UNTIL' N-2 'DØ'
          U[I]←W[I]←B[I];
          'FØR' I←3 'STEP' 1 'UNTIL' N-1 'DØ'
          T[I]←X[I]←E[I];
          Q2[1]←S[1]←A1;
          X[N-2]←X[N-1]←W[N-2]←V[N-2]←V[N-1]←V[N]←0;
          Q3[1]←R[1]←A2;
          Q3[2]←R[2]←A3;
          Q2[N]←Q3[N-1]←B[N];
          Q1[N-1]←Q2[N-2]←B[N-1];
          Q1[N-3]←B[N-2];
          Q3[N]←C[N]-EIG[M];
          Q2[N-1]←C[N-1]-EIG[M];
          Q1[N-2]←C[N-2]-EIG[M];
          Q1[N]←Q3[N-2]←E[N];
          Q2[N-3]←T[N-1]←E[N-1];
```

```
Q1[N-4]←E[N-2];
S[N-3]←E[N-1];
'CØMMENT'  Now the elimination process can be performed in
     the first N-4 columns
'FØR' I←1 'STEP' 1 'UNTIL' N-4 'DØ'
'BEGIN'
     D3←ABS(R[I]);
     D4←ABS(V[I]);
     D←ABS(U[I+1]);
     D1←ABS(T[I+2]);
     D2←ABS(S[I]);
     'IF' D 'GT' D1
     'AND' D 'GT' D3
     'AND' D 'GT' D4
     'THEN'
     'BEGIN'
          IC[I]←I+1;
          D←V[I]; V[I]←U[I+1]; U[I+1]←D;
          D←W[I+1]; W[I+1]←V[I+1]; V[I+1]←D;
          D←X[I+2]; X[I+2]←W[I+2]; W[I+2]←D;
          D←Y[I+3]; Y[I+3]←X[I+3]; X[I+3]←D;
          D←Z[I+4]; Z[I+4]←Y[I+4]; Y[I+4]←D;
          D←Q1[I]; Q1[I]←Q1[I+1]; Q1[I+1]←D;
          D←Q2[I]; Q2[I]←Q2[I+1]; Q2[I+1]←D;
          D←Q3[I]; Q3[I]←Q3[I+1]; Q3[I+1]←D;
     'END'
     'ELSE'
     'IF' D1 'GT' D2
     'AND' D1 'GT' D3
     'AND' D1 'GT' D4
     'THEN'
     'BEGIN'
          IC[I]←I+2;
          D←V[I]; V[I]←T[I+2]; T[I+2]←D;
          D←W[I+1]; W[I+1]←U[I+2]; U[I+2]←D;
          D←X[I+2]; X[I+2]←V[I+2]; V[I+2]←D;
          D←Y[I+3]; Y[I+3]←W[I+3]; W[I+3]←D;
          D←Z[I+4]; Z[I+4] X[I+4]; X[I+4]←D;
          D←Q1[I]; Q1[I]←Q1[I+2]; Q1[I+2]←D;
          D←Q2[I]; Q2[I]←Q2[I+2]; Q2[I+2]←D;
          D←Q3[I]; Q3[I]←Q3[I+2]; Q3[I+2]←D;
     'END'
     'ELSE'
     'IF' D2 'GT' D3
     'AND' D2 'GT' D4
     'THEN'
     'BEGIN'
          D←V[I]; V[I]←S[I]; S[I]←D;
          D←W[I+1]; W[I+1]←S[I+1]; S[I+1]←D;
          D←X[I+2]; X[I+2]←S[I+2]; S[I+2]←D;
          D←Y[I+3]; Y[I+3]←S[I+3]; S[I+3]←D;
          D←X[I+4]; Z[I+4]←S[I+4]; S[I+4]←D;
          D←Q1[I]; Q1[I]←Q1[N-1]; Q1[N-1]←D;
          D←Q2[I]; Q2[I]←Q2[N-1]; Q2[N-1]←D;
          D←Q3[I]; Q3[I]←Q3[N-1]; Q3[N-1]←D;
          IC[I]←N-1;
     'END'
```

```
          'ELSE'
          'IF' D3 'GT' D4
          'THEN'
          'BEGIN'
               D←V[I]; V[I]←R[I]; R[I]←D;
               D←W[I+1]; W[I+1]←R[I+1]; R[I+1]←D;
               D←X[I+2]; X[I+2]←R[I+2]; R[I+2]←D;
               D←Y[I+3]; Y[I+3]←R[I+3]; R[I+3]←D;
               D←Z[I+4]; Z[I+4]←R[I+4]; R[I+4]←D;
               D←Q1[I]; Q1[I]←Q1[N]; Q1[N]←D;
               D←Q2[I]; Q2[I]←Q2[N]; Q2[N]←D;
               D←Q3[I]; Q3[I]←Q3[N]; Q3[N]←D;
               IC[I]←N;
          'END';
          U[I]←U[I+1]/V[I];
          T[I]←T[I+2]/V[I];
          S[I]←S[I]/V[I];
          R[I]←R[I]/V[I];
          V[I+1]←V[I+1]-W[I+1]*U[I];
          W[I+2]←W[I+2]-X[I+2]*U[I];
          X[I+3]←X[I+3]-Y[I+3]*U[I];
          Y[I+4]←Y[I+4]-Z[I+4]*U[I];
          U[I+2]←U[I+2]-W[I+1]*T[I];
          V[I+2]←V[I+2]-X[I+2]*T[I];
          W[I+3]←W[I+3]-Y[I+3]*T[I];
          X[I+4]←X[I+4]-Z[I+4]*T[I];
          S[I+1]←S[I+1]-W[I+1]*S[I];
          S[I+2]←S[I+2]-X[I+2]*S[I];
          S[I+3]←S[I+3]-Y[I+3]*S[I];
          S[I+4]←S[I+4]-Z[I+4]*S[I];
          R[I+1]←R[I+1]-W[I+1]*R[I];
          R[I+2]←R[I+2]-X[I+2]*R[I];
          R[I+3]←R[I+3]-Y[I+3]*R[I];
          R[I+4]←R[I+4]-Z[I+4]*R[I];
          Q1[I+1]←Q1[I+1]-Q1[I]*U[I];
          Q1[I+2]←Q1[I+2]-Q1[I]*T[I];
          Q1[N-1]←Q1[N-1]-Q1[I]*S[I];
          Q1[N]←Q1[N]-Q1[I]*R[I];
          Q2[I+1]←Q2[I+1]-Q2[I]*U[I];
          Q2[I+2]←Q2[I+2]-Q2[I]*T[I];
          Q2[N]←Q2[N]-Q2[I]*R[I];
          Q2[N-1]←Q2[N-1]-Q2[I]*S[I];
          Q3[I+1]←Q3[I+1]-Q3[I]*U[I];
          Q3[I+2]←Q3[I+2]-Q3[I]*T[I];
          Q3[N-1]←Q3[N-1]-Q3[I]*S[I];
          Q3[N]←Q3[N]-Q3[I]*R[I];
     'END';
     'COMMENT' The elimination in the last four columns now
          has to be performed seperately; First in column (N-3);
     T[N-1]←S[N-3];
     D←ABS(U[N-2]);
     D1←ABS(T[N-1]);
     D2←ABS(R[N-3]);
     D3←ABS(V[N-3]);
     'IF' D 'GT' D1
     'AND' D 'GT' D2
     'AND' D 'GT' D3
     'THEN'
```

```
'BEGIN'
     D←V[N-3]; V[N-3]←U[N-2]; U[N-2]←D;
     D←Q1[N-3]; Q1[N-3]←Q1[N-2]; Q1[N-2]←D;
     D←Q2[N-3]; Q2[N-3]←Q2[N-2]; Q2[N-2]←D;
     D←Q3[N-3]; Q3[N-3]←Q3[N-2]; Q3[N-2]←D;
     IC[N-3]←N-2;
'END'
'ELSE'
'IF' D1 'GT' D2
'AND' D1 'GT' D3
'THEN'
'BEGIN'
     IC[N-3]←N-1;
     D←V[N-3]; V[N-3]←T[N-1]; T[N-1]←D;
     D←Q1[N-3]; Q1[N-3]←Q1[N-1]; Q1[N-1]←D;
     D←Q2[N-3]; Q2[N-3]←Q2[N-1]; Q2[N-1]←D;
     D←Q3[N-3]; Q3[N-3]←Q3[N-1]; Q3[N-1]←D;
'END'
'ELSE'
'IF' D2 'GT' D3
'THEN'
'BEGIN'
     D←V[N-3]; V[N-3]←R[N-3]; R[N-3]←D;
     D←Q1[N-3]; Q1[N-3]←Q1[N]; Q1[N]←D;
     D←Q2[N-3]; Q2[N-3]←Q2[N]; Q2[N]←D;
     D←Q3[N-3]; Q3[N-3]←Q3[N]; Q3[N]←D;
     IC[N-3]←N;
'END';
U[N-3]←U[N-2]/V[N-3];
T[N-3]←T[N-1]/V[N-3];
R[N-3]←R[N-3]/V[N-3];
Q1[N-2]←Q1[N-2]-U[N-3]*Q1[N-3];
Q2[N-2]←Q2[N-2]-U[N-3]*Q2[N-3];
Q3[N-2]←Q3[N-2]-U[N-3]*Q3[N-3];
Q1[N-1]←Q1[N-1]-Q[N-3]*T[N-3];
Q2[N-1]←Q2[N-1]-Q2[N-3]*T[N-3];
Q3[N-1]←Q3[N-1]-Q3[N-3]*T[N-3];
Q1[N]←Q1[N]-Q1[N-3]*R[N-3];
Q2[N]←Q2[N]-Q2[N-3]*R[N-3];
Q3[N]←Q3[N]-Q3[N-3]*R[N-3];
'CØMMENT' Elimination is now performed in column N-2;
D←ABS(Q1[N-1]); D1←ABS(Q1[N]);
D2←ABS(Q1[N-2]);
'IF' D 'GT' D1
'AND' D 'GT' D2
'THEN'
'BEGIN'
     D←Q1[N-2]; Q1[N-2]←Q1[N-1]; Q1[N-1]←D;
     D←Q2[N-2]; Q2[N-2]←Q2[N-1]; Q2[N-1]←D;
     D←Q3[N-2]; Q3[N-2]←Q3[N-1]; Q3[N-1]←D;
     IC[N-2]←N-1;
'END'
'IF' D1 'GT' D2
'THEN'
'BEGIN'
     D←Q1[N-2]; Q1[N-2]←Q1[N]; Q1[N]←D;
```

```
     D←Q2[N-2]; Q2[N-2]←Q2[N]; Q2[N]←D;
     D←Q3[N-2]; Q3[N-2]←Q3[N]; Q3[N]←D;
     IC[N-2]←N;
'END';
Q1[N-1]←Q1[N-1]/Q1[N-2];
Q1[N]←Q1[N]/Q1[N-2];
Q2[N-1]←Q2[N-1]-Q1[N-1]*Q2[N-2];
Q3[N-1]←Q3[N-1]-Q3[N-2]*Q1[N-1];
Q2[N]←Q2[N]-Q1[N]*Q2[N-2];
Q3[N]←Q3[N]-Q1[N]*Q3[N-2];
'CØMMENT' Elimination is finally performed in column N-1;
'IF' ABS(Q2[N]) 'GT' ABS(Q2[N-1])
'THEN'
'BEGIN'
     D←Q2[N]; Q2[N]←Q2[N-1]; Q2[N-1]←D;
     D←Q3[N]; Q3[N]←Q3[N-1]; Q3[N-1]←D;
     IC[N-1]←N;
'END';
Q2[N]←Q2[N]/Q2[N-1];
Q3[N]←Q3[N]-Q2[N]*Q3[N-1];
'IF' Q3[N] 'EQ' 0 'THEN' Q3[N]←0.0000000001;
'CØMMENT' In the decomposing case Q3[N] is replaced to avoid
     division by zero. The initial back substitution now
     takes place;
Q[N]←Q[N]/Q3[N];
Q[N-1]←(Q[N-1]-Q3[N-1]*Q[N])/Q2[N-1];
Q[N-2]←(Q[N-2]-Q3[N-2]*Q[N]-Q2[N-2]*Q[N-1])/Q1[N-2];
Q[N-3]←(Q[N-3]-Q3[N-3]*Q[N]-Q2[N-3]*Q[N-1]-Q1[N-3]*Q[N-2])/V[N-3];
'FØR' I←N-4 'STEP' -1 'UNTIL' 1 'DØ'
Q[I]←(Q[I]-Q3[I]*Q[N]-Q2[I]*Q[N-1]-Q1[I]*Q[N-2]
     -W[I+1]*Q[I+1]-X[I+2]*Q[I+2]-Y[I+3]*Q[I+3]-
     Z[I+4]*Q[I+4])/V[I];
'CØMMENT' Elimination with improved eigenvector estimate is
     now performed. This time the stored interchanges and
     elimination factors only are used;
'FØR' I←1 'STEP' 1 'UNTIL' N-4 'DØ'
'BEGIN'
     'IF' IC[I] 'NE' 0 'THEN'
     'BEGIN'
          D←Q[I]; Q[I]←Q[IC[I]]; Q[IC[I]]←D;
     'END';
     Q[I+1]←Q[I+1]-U[I]*Q[I];
     Q[I+2]←Q[I+2]-T[I]*Q[I];
     Q[N-1]←Q[N-1]-S[I]*Q[I];
     Q[N]←Q[N]-R[I]*Q[I];
'END';
'IF' IC[N-3] 'NE' 0 'THEN'
'THEN'
'BEGIN'
     D←Q[N-3]; Q[N-3]←Q[IC[N-3]]; Q[IC[N-3]]←D;
'END';
Q[N-2]←Q[N-2]-Q[N-3]*U[N-3];
Q[N-1]←Q[N-1]-Q[N-3]*T[N-3];
Q[N]←QN-Q[N-3]*R[N-3];
'IF' IC[N-2] 'NE' 0 'THEN'
```

```
            'BEGIN'
                D←Q[N-2]; Q[N-2]←Q[IC[N-2]]; Q[IC[N-2]]←D;
            'END';
            Q[N-1]←Q[N-1]-Q[N-2]*Q1[N-1];
            Q[N]←Q[N]-Q[N-2]*Q1[N];
            'IF' IC[N-1] 'NE' 0 'THEN'
            'BEGIN'
                D←Q[N-1]; Q[N-1]←Q[N]; Q[N]←D;
            'END';
            Q[N]←Q[N]-Q[N-1]*Q2[N];
            'CØMMENT' Back substitution is now performed for the final
                time;
            Q[N]←Q[N]/Q3[N];
            Q[N-1]←(Q[N-1]-Q3[N-1]*Q[N])/Q2[N-1];
            Q[N-2]←(Q[N-2]-Q3[N-2]*Q[N]-Q2[N-2]*Q[N-1])/Q1[N-2];
            Q[N-3]←(Q[N-3]-Q3[N-3]*Q[N]-Q2[N-3]*Q[N-1]-
                Q1[N-3]*Q[N-2])/V[N-3];
            'FØR' I←N-4 'STEP' -1 'UNTIL' 1 'DØ'
            Q[I]←(Q[I]-Q3[I]*Q[N]-Q2[I]*Q[N-1]-Q1[I]*Q[N-2]
                -W[I+1]*Q[I+1]-X[I+2]*Q[I+2]-Y[I+3]*Q[I+3]
                -Z[I+4]*[[I+4])/V[I];
            'CØMMENT' Now the eigenvector is normalised and stored in the
                appropriate column of VEC;
            D←Q[1]*Q[1];
            'FØR' I←2 'STEP' 1 'UNTIL' N 'DØ'
            D←D+Q[I]*Q[I];
            D←1/SQRT(D);
            'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
            VEC[I,M]←Q[I]*D;
            'CØMMENT' The Rayleigh Quotient is calculated to improve the
                estimate of the eigenvalue;
            D1←(C[1]*VEC[1,M]+B[2]*VEC[2,M]+E[3]*VEC[3,M]
                +A1*VEC[N-1,M]+A2*VEC[N,M])*VEC[1,M];
            D1←D1+(B[2]*VEC[1,M]+C[2]*VEC[2,M]+B[3]*
                VEC[3,M]+E[4]*VEC[4,M]+A3*VEC[N,M])*VEC[2,M];
            'FØR' I←3 'STEP' 1 'UNTIL N-2 'DØ'
            D1←D1+(C[I]*VEC[I,M]+B[I]*VEC[I-1,M]+E[I]*VEC[I-2,M]
                +B[I+1]*VEC[I+1,M]+E[I+2]*VEC[I+2,M])*VEC[I,M];
            D1←D1+(C[N-1]*VEC[N-1,M]+B[N]*VEC[N,M]+B[N-1]*VEC[N-2,M]
                E[N-1]*VEC[N-3 ,M]+A1*VEC[1,M])*VEC[N-1,M];
            D1←D1+(C[N]*VEC[N,M]+B[N]*VEC[N-1,M]+E[N]*VEC[N-2,M]
                +A3*VEC[2,M]+A2*VEC[1,M])*VEC[N,M];  EIG(M)←D1;
        'END';
'END';
```

PROGRAM 18

This program determines the eigenvalues of a symmetric matrix using the modified bisection algorithm of Chapter 5. The matrix used is septdiagonal with a band at semi-bandwidth P, and quindiagonal in the centre.

```
'PRØCEDURE' SDEMB(C,B,D,E,N,N1,N2,P,EPS,QZR,EIG);
'CØMMENT' C is the diagonal, B, the sub-diagonal, D the sub-sub-
    diagonal and E is the band at semi-bandwidth P of the matrix of
    order N. N1 is the number of the smallest eigenvalue required
    and N2 the largest. EPS is the required accuracy and QZR the
    closest number to machine zero (substituted to avoid zero
    divisions). The eigenvalues are stored in the first (N2-N1+1)
    elements of EIG;
'ARRAY' C,B,D,E,EIG;
'INTEGER' N,P,N1,N2;
'REAL' EPS,QZR;
'BEGIN'
    'ARRAY' W,U,V,[1:N],R[P:N,4:P];
    'REAL' MAX,MIN,NEW,NEWER,LAMBDA,S;
    'INTEGER' I,J,K,L,M,T;
    'BEGIN'
        'CØMMENT' Bounds for the whole eigenvalue range are now
            determined (MAX,MIN);
        NEW←ABS(B[N])+ABS(D[N])+ABS(E[N]);
        B[1]←E[2]←E[1]←0;
        MAX←C[N]+NEW;
        MIN←C[N]-NEW;
        'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
        'BEGIN'
            NEW←ABS(B[I])+ABS(B[I+1])+ABS(E[I])+('IF' I+2
                'GT' N 'THEN' 0 'ELSE' ABS(E[I+2]));
            'IF' P-1+I 'LE' N 'THEN'
            NEW←NEW+ABS(D[P-1+I]);
            'IF' I 'GE' P 'THEN'
            NEW←NEW+ABS(D[I]);
            NEWER←C[I]+NEW;
            NEW←C[I]-NEW;
            'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
            'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'END';
    'END';
    'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
    R[I,P]←D[I];
    'FØR' I←3 'STEP' 1 'UNTIL' P-1 'DØ'
    W[I]←E[I];
    V[1]←W[2]←W[1]←0;
    V[2]←B[2];
    L←N1-1;
LABA: 'IF' L 'GE' N2 'THEN' 'GOTO' EXIT;
```

```
      'CØMMENT' Major loop for the L^th eigenvalue;
      LAMBDA←(MIN+MAX)*0.5;
LABB: 'BEGIN'
          'CØMMENT' Calculation of the Sturm sequence;
          U[1]←C[1]-LAMBDA;
          M←0;
          'IF' U[1] 'LT' 0 'THEN' M←M+1;
          'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
          'IF' U[1] 'EQ' 0 'THEN' U[1]←QZR;
          U[2]←C[2]-LAMBDA-B[2]*B[2]/U[1];
          'IF' U[2] 'LT' 0 'THEN' M←M+1;
          'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
          'IF' U[2] 'EQ' 0 'THEN' U[2]←QZR;
          'FØR' I←3 'STEP' 1 'UNTIL' P-1 'DØ'
          'BEGIN'
              V[I]←B[I]-V[I-1]*W[I]/U[I-2];
              U[I]←C[I]-LAMBDA-V[I]*V[I]/U[I-1]-W[I]*W[I]/U[I-2];
              'IF' U[I] 'LT' 0'THEN' M←M+1;
              'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
              'IF' U[I] 'EQ' 0 'THEN' U[I]←QZR;
          'END';
          'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
          'BEGIN'
              'FØR' K←P-1 'STEP' -1 'UNTIL' 4 'DØ'
              'BEGIN'
                  T←I-K+1;
                  S←-R[I,K+1]*V[T]/U[T-1];
                  'IF' K+2 'LE' P 'THEN'
                  S←S-R[I,K+2]*W[T]/U[T-2];
                  'IF' T 'GE' P 'THEN'
                  'FOR' J←P-K-3 'STEP' -1 'UNTIL' 0 'DØ'
                  S←S-R[I,P-J]*R[T,P+1-K-J]/U[I-P+J+1];
                  R[I,K]←S;
              'END';
              S←E[I]-S*V[I-2]/U[I-3]-R[I,5]*W[I-2]/U[I-4];
              'IF' I-2 'GE' P 'THEN'
              'FØR' J←P-6 'STEP' -1 'UNTIL' 0 'DØ'
              S←S-R[I,P-J]*R[I-2,P-2-J]/U[I-P+J+1];
              W[I]←S;
              S←B[I]-S*V[I-1]/U[I-2]-R[I,4]*W[I-1]/U[I-3];
              'IF' I-1 'GE' P 'THEN'
              'FØR' J←P-5 'STEP' -1 'UNTIL' 0 'DØ'
              S←S-R[I,P-J]*R[I-1,P-1-J]/U[I-P+J+1];
              V[I]←S;
              S←C[I]-LAMBDA-S*S/U[I-1]-W[I]*W[I]/U[I-2];
              'FØR' J←P-4 'STEP' -1 'UNTIL' 0 'DØ'
              S←S-R[I,P-J]*R[I,P-J]/U[I-P+J+1];
              'IF' S 'LT' 0 'THEN' M←M+1;
              'IF' L-M 'LT' 0 'ØR' L-M-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
              'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
          'END';
      'END';
 LABC: 'IF' L-M 'GE' 0 'THEN'
      'BEGIN'
          NEWER←MIN;
          MIN←LAMBDA;
          LAMBDA←LAMBDA+(LAMBDA-NEWER)*0.5;
      'END'
```

```
        'ELSE'
        'BEGIN'
             LAMBDA←(LAMBDA+MIN)*0.5;
             'IF' (LAMBDA-MIN)*0.5/ABS(LAMBDA) 'LT' EPS 'THEN'
             'BEGIN'
                  MIN←2 *LAMBDA-MIN;
                  'FØR' I←L+1 'STEP' 1 'UNTIL'
                  ('IF' M 'GT' N2 'THEN' N2 'ELSE' M) 'DØ'
                  EIG[I-N1+1]←LAMBDA;
                  L←M;
                  'GØTØ' LABA;
             'END'
      'END'; 'GOTO' LABB;
EXIT: 'END';
```

PROGRAM 19

This program determines the eigenvalues of a symmetric sparse banded matrix using the modified bisection algorithm of Chapter 5. The matrix for this algorithm has bands between semi-bandwidth P and M and is tridiagonal in the centre.

```
'PRØCEDURE' SBBTMB(C,B,D,N,N1,N2,P,M,EPS,QZR,EIG);
'CØMMENT' C is the diagonal, B the sub-diagonal, and D the bands at
    semi-bandwidth P to M of the Nth order matrix. N1 is the number
    of the smallest eigenvalue required, and N2 the largest. EPS is
    the required accuracy and QZR the closest number to machine zero
    (substituted to avoid zero divisions). The eigenvalues are stored
    in the first (N2-N1+1) elements of EIG;
'ARRAY' C,B,D,EIG;
'INTEGER' N,P,M,N1,N2;
'REAL' EPS, QZR;
'BEGIN'
    'ARRAY' R[P:N,3:M],U,V[1:N];
    'REAL' MAX,MIN,NEW,NEWER,LAMBDA,S;
    'INTEGER' I,J,K,L,Z,T;
    'BEGIN'
        'CØMMENT' The bounds on the complete eigenvalue range are
            determined (MAX,MIN);
        B[1]←0;
        NEW←ABS(B[N]);
        'FØR' I←P 'STEP' 1 'UNTIL' M 'DØ'
        NEW←NEW+ABS(D[N,I]);
        MAX←C[N]+NEW;
        MIN←C[N]-NEW;
        'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
        'BEGIN'
            NEW←ABS(B[I])+ABS(B[I+1]);
            'FØR' J←P 'STEP' 1 'UNTIL'
            'IF' M-1+I 'LE' N 'THEN' M
            'ELSE' M+M-I-P 'DØ'
            NEW←NEW+ABS(D[I+J-1,J]);
            'FØR' J 'IF' I 'LT' M 'THEN' I 'ELSE' M
            'STEP' -1 'UNTIL' P 'DØ'
            NEW←NEW+ABS(D[I,J]);
            NEWER←C[I]+NEW;
            NEW←C[I]-NEW;
            'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
            'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'END';
    'END';
    'CØMMENT' The elements of the Sturm sequence are now initialised;
    'FØR' I←P 'STEP' 1 'UNTIL' M 'DØ'
    R[I,I]←D[I,I];
    'FØR' I←M 'STEP' 1 'UNTIL' N 'DØ'
    R[I,M]←D[I,M];
    'FØR' I←1 'STEP' 1 'UNTIL' P-1 'DØ'
```

```
        V[I]←B[I];
        L←N1-1;
LABA:   'IF' L 'GE' N2 'THEN' 'GØTØ' EXIT;
        'CØMMENT' Major loop for the Lth eigenvalue;
        LAMBDA←(MIN+MAX)*0.5;
LABB:   'BEGIN'
            'CØMMENT' Calculation of the Sturm sequence;
            U[1]←C[1]-LAMBDA;
            Z←0;
            'IF' U[1] 'LT' 0 'THEN' Z←Z+1;
            'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
            'IF' U[1] 'EQ' 0 'THEN' U[1]←QZR;
            'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
            'BEGIN'
                U[I]←C[I]-LAMBDA-V[I]*V[I]/U[I-1];
                'IF' U[I] 'LT' 0 'THEN' Z←Z+1;
                'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
                'IF' U[I] 'EQ' 0 'THEN' U[I]←QZR;
            'END';
            'FOR' I←P 'STEP' 1 'UNTIL' M 'DO'
            'BEGIN'
                R[I,I-1]←-R[I,I]*V[2]/U[1];
                'FOR' K←I-2 'STEP' -1 'UNTIL' 3 'DO'
                'BEGIN'
                    T←I-K+1;
                    S←-R[I,K+1]*V[T]/U[T-1];
                    'IF' T 'GE' P 'THEN'
                    'FØR' J←I-K-2 'STEP' -1 'UNTIL' 0 'DØ'
                    S←S-R[I,I-J]*R[T,T-J]/U[J+1];
                    'IF' K 'GE' P 'THEN' S←S+D[I,K];
                    R[I,K]←S;
                'END';
                S←B[I]-S*V[I-1]/U[I-2];
                'IF' I-1 'GE' P 'THEN'
                'FØR' J←I-4 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,I-J]*R[I-1,I-1-J]/U[J+1];
                V[I]←S;
                S←C[I]-LAMBDA-V[I]*V[I]/U[I-1];
                'FØR' J←I-3 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,I-J]*R[I,I-J]/U[J+1];
                'IF' S 'LT' 0 'THEN' Z←Z+1;
                'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
                'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE] U[I]←QZR;
            'END';
            'FØR' I←M+1 'STEP' 1 'UNTIL' N 'DØ'
            'BEGIN'
                R[I,M-1]←-R[I,M]*V[I-M+2]/U[I-M+1];
                'FØR' K←M-2 'STEP' -1 'UNTIL' 3 'DØ'
                'BEGIN'
                    T←I-K+1;
                    S←-R[I,K+1]*V[T]/U[T-1];
                    'IF' T 'GE' P 'THEN'
                    'FØR' J←M-K-2 'STEP' -1 ' UNTIL' 0 'DØ'
                    S←S-R[I,M-J]*R[T,M+1-K-J]/U[I-M+J+1];
                    'IF' K 'GE' P 'THEN' S←S+D[I,K];
                    R[I,K]←S;
                'END';
                S←B[I]-S*V[I-1]/U[I-2];
```

```
                'IF' I-1 'GE' P 'THEN'
                'FØR' J←M-4 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,M-J]*R[I-1,M-1-J]/U[I-M+J+1];
                V[I]←S;
                S←C[I]-LAMBDA-V[I]*V[I]/U[I-1];
                'FØR' J←M-3 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,M-J]*R[I,M-J]/U[I-M+J+1];
                'IF' S 'LT' 0 'THEN' Z←Z+1;
                'IF' L-Z 'LT' 0 'ØR' L-Z-M+I'GE' 0 'THEN' 'GØTØ' LABC;
                'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
           'END';
     'END';
LABC:'IF' L-Z 'GE' 0 'THEN'
     'BEGIN'
          NEWER←MIN;
          MIN←LAMBDA;
          LAMBDA LAMBDA+(LAMBDA-NEWER)*0.5;
     'END'
     'ELSE'
     'BEGIN'
          LAMBDA←(LAMBDA+MIN)*0.5;
          'IF' (LAMBDA-MIN)*0.5/ABS(LAMBDA) 'LT' EPS 'THEN'
          'BEGIN'
               MIN←2*LAMBDA-MIN;
               'FOR' I←L+1 'STEP' 1 'UNTIL'
               ('IF' Z 'GT' N2 'THEN' N2 'ELSE'Z ) 'DØ'
               EIG[I-N1+1]←LAMBDA;
               L←Z;
               'GØTØ' LABA;
          'END';
     'END';
     'GØTØ'LABB;
 EXIT:
 'END';
```

PROGRAM 20

This program determines the eigenvalues of a symmetric sparse banded matrix using the modified bisection algorithm of Chapter 5. The matrix for this algorithm has bands between semi-bandwidth P and M, and is quindiagonal in the centre.

```
'PRØCEDURE SBBQMB(C,B,D,E,N,N1,N2,P,M,EPS,QZR,EIG);
'CØMMENT' C is the diagonal, B the sub-diagonal, E the sub-sub-diagonal,
    and D the bands at semi-bandwidth P to M of the (N×N) matrix.  N1 is
    the number of the smallest eigenvalue required and N2 the largest.
    EPS is the required accuracy and QZR the closest number to machine
    zero (substituted to avoid zero divisions).  The eigenvalues are
    stored in the first (N2-N1+1) elements of EIG;
'ARRAY' C,B,D,E,EIG;
'INTEGER' N,P,N1,N2,M;
'REAL' EPS, QZR;
'BEGIN'
    'ARRAY' R[P:N,4:M],V,U,W[1:N];
    'REAL' MAX,MIN,NEW,NEWER,LAMBDA,S;
    'INTEGER' I,J,K,L,Z,T;
    'BEGIN'
        'CØMMENT' Bounds for the complete eigenvalue range are
            determined (MAX,MIN);
        B[1]←E[1]←E[2]←0;
        NEW←ABS(B[N])+ABS[E[N]);
        'FØR' I←P 'STEP' 1 'UNTIL' M 'DØ'
        NEW←NEW+ABS(D[N,I]);
        MAX←C[N]+NEW;
        MIN←C[N]-NEW;
        'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
        'BEGIN'
            NEW←ABS(B[I])+ABS(B[I+1])+ABS(E[I])+
                ('IF' I 'NE' N-1 'THEN' ABS(E[I+2])'ELSE' 0);
            'FØR' J←P 'STEP' 1 'UNTIL' 'IF' M-1+I 'LE'
                N 'THEN' M 'ELSE' M+M-I-P 'DØ'
            NEW←NEW+ABS(D[I+J-1,J]);
            'FØR' 'IF' I 'LT' M 'THEN' I 'ELSE' M
                'STEP' -1 'UNTIL' P 'DØ'
            NEW←NEW+ABS(D[I,J]);
            NEWER←C[I]+NEW;
            NEW←C[I]-NEW;
            'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
            'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
        'END';
    'END';
    'FØR' I←P 'STEP' 1 'UNTIL' M 'DØ'
    R[I,I]←D[I,I];
    'FØR' I←M 'STEP' 1 'UNTIL' N 'DØ'
    R[I,M]←D[I,M];
    'FØR' I←3 'STEP' 1 'UNTIL' P-1 'DØ'
    W[I]←E[I];
```

```
      V[1]←W[1]←W[2]←0;
      V[2]←B[2];
      L←N1-1;
LABA: 'IF' L 'GE' N2 'THEN' 'GØTØ' EXIT;
      LAMBDA←(MIN+MAX)*0.5;
LABB: 'BEGIN'
           Z←0;
           U[1]←C[1]-LAMBDA;
           'IF' U[1] 'LT' 0 'THEN' Z←Z+1;
           'IF' L-Z 'LT' 0 'ØR' L-Z-N+I'GE' 0 'THEN' ' GØTØ' LABC;
           'IF' U[1] 'EQ' 0 'THEN' U[1]←QZR;
           U[2]←C[2]-LAMBDA-V[2]*V[2]/U[1];
           'IF' U[2] 'LT' 0 'THEN' Z←Z+1;
           'IF' L-Z 'LT' 0 'ØR' L-Z-N+I'GE' 0 'THEN' 'GØTØ' LABC;
           'IF' U[2] 'EQ' 0 'THEN' Z←Z+1;
           'FØR' I←3 'STEP' 1 'UNTIL' P-1 'DØ'
           'BEGIN'
                V[I]←B[I]-V[I-1]*W[I]/U[I-2];
                U[I]←C[I]-LAMBDA-V[I]*V[I]/U[I-1]-W[I]*W[I]/U[I-2];
                'IF' U[I] 'LT' 0 'THEN' Z←Z+1;
               'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
           'END';
           'FØR' I←P 'STEP' 1 'UNTIL' M 'DØ'
           'BEGIN'
                S←R[I,I-1] -R[I,I]*V[2]/U[I];
                S←R[I,I-2] -S*V[3]/U[2]-R[I,I]*W[3]/U[1];
                'FØR' K←I-3 'STEP' -1 'UNTIL' 4 'DØ'
                'BEGIN'
                     T←I-K+1
                     S←-S*V[T]/U[T-1]-R[I,K+2]*W[T]/U[T-2];
                     'IF' T 'GE' P 'THEN'
                     'FØR' J←I-K-3 'STEP' -1 'UNTIL' 0 'DØ'
                     S←S-R[I,I-J]*R[T,T-J]/U[J+1];
                     'IF' K 'GE' P 'THEN' S←S+D[I,K];
                     R[I,K]←S;
                'END';
                S←E[I]-S*V[I-2]/U[I-3]-R[I,5]*W[I-2]/U[I-4];
                'IF' I-2 'GE' P 'THEN'
                'FØR' J←I-6 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,I-J]*R[I-2,I-2-J]/U[J+1];
                W[I]←S;
                S←B[I]-S*V[I-1]/U[I-2]-R[I,4]*W[I-1]/U[I-3];
                'IF' I-1 'GE' P 'THEN'
                'FØR' J←I-5 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,I-J]*R[I-1,I-1-J]/U[J+1];
                V[I]←S;
                S←C[I]-LAMBDA-S*S/U[I-1]-W[I]*W[I]/U[I-2];
                'FØR' J←I-4 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,I-J]*R[I,I-J]/U[I+1];
                'IF' S 'LT' 0 'THEN' Z←Z+1;
                'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GØTØ' LABC;
                'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
           'END';
```

```
            'FØR' I←M+1 'STEP' 1 'UNTIL' N 'DØ'
            'BEGIN'
                S←R[I,M-1]←-R[I,M]*V[I-M+2]/U[I-M+1];
                S←R[I,M-2]←-S*V[I-M+3]/U[I-M+2]-R[I,M]*W[I-M+3]/U[I-M+1];
                'FØR' K←M-3 'STEP' -1 'UNTIL' 4 'DØ'
                'BEGIN'
                    T←I-K+1;
                    S←-S*V[I]/U[T-1]-R[I,K+2]*W[T]/U[T-2];
                    'IF' T 'GE' P 'THEN'
                    'FØR' J←M-K-3 'STEP' -1 'UNTIL' 0 'DØ'
                    S←S-R[I,M-J]*R[T,M+1-K-J]/U[I-M+J+1];
                    'IF' K 'GE' P 'THEN' S←S+D[I,K];
                    R[I,K]←S;
                'END';
                S←E[I]-S*V[I-2]/U[I-3]-R[I,5]*W[I-2]/U[I-4];
                'IF' I-2 'GE' P 'THEN'
                'FØR' J←M-6 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,M-J]*R[I-2,M-2-J]/U[I-M+J+1];
                W[I]←S;
                S←B[I]-S*V[I-1]/U[I-2]-R[I,4]*W[I-1]/U[I-3];
                'IF' I-1 'GE' P 'THEN'
                'FØR' J←M-5 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,M-J]*R[I-1,M-1-J]/U[I-M+J+1];
                V[I]←S;
                S←C[I]-LAMBDA-S*S/U[I-1]-W[I]*W[I]/U[I-2];
                'FØR' J←M-4 'STEP' -1 'UNTIL' 0 'DØ'
                S←S-R[I,M-J]*R[I,M-J]/U[I-M+J+1];
                'IF' S 'LT' 0 'THEN' Z←Z+1;
                'IF' L-Z 'LT' 0 'ØR' L-Z-N+I 'GE' 0 'THEN' 'GOTO' LABC;
                'IF' S 'NE' 0 'THEN' U[I]←S 'ELSE' U[I]←QZR;
            'END';
        'END';
LABC:   'IF' L-Z 'GE' 0 'THEN'
        'BEGIN'
            NEWER←MIN;
            MIN←LAMBDA;
            LAMBDA←LAMBDA+(LAMBDA-NEWER)*0.5;
        'END'
        'ELSE'
        'BEGIN'
            LAMBDA←(LAMBDA+MIN)*0.5;
            'IF' (LAMBDA-MIN)*0.5/ABS(LAMBDA) 'LT' EPS 'THEN'
            'BEGIN'
                MIN←2*LAMBDA-MIN;
                'FØR' I←L+1 'STEP' 1 'UNTIL'
                ('IF' Z 'GT' N2 'THEN' N2 'ELSE' Z) 'DØ'
                EIG[I-N1+1]←LAMBDA;
                L←Z;
                'GØTØ' LABA;
            'END';
         'END';
         'GØTØ' LABB;
EXIT:
'END';
```

PROGRAM 21

This program performs the Partial Secant method on symmetric banded quindiagonal matrices. The secand method is modified by the use of Partial Sturm Sequences, to speed it up, and the method proposed by Anderson (1975) to determine eigenvalues in ascending or descending order. The program has been left in a 'rough' form as improvements are still being made, and in this form it is easier to follow. It is set up to determine eigenvalues starting with the minimum eigenvalue, but can be easily changed to determine the largest in descending order.

```
'PRØCEDURE' PSS(C,B,D,N,P,N1,EPS,QZR,EIG);
'CØMMENT' C is the main diagonal, B the sub-diagonal, and D the diagonal
     at semi-bandwidth P of the (N×N) symmetric matrix. EPS is the
     accuracy required, and QZR the number closest to machine zero,
     used to avoid zero divisions N1 eigenvalues are required starting
     with the smallest, and are stored in the first N1 elements of EIG;
'ARRAY' C,B,D,EIG;
'REAL' EPS,QZR;
'INTEGER' N,P,N1;
'BEGIN'
     'ARRAY' R[P:N,3:P],U,V,S[0:N];
     'INTEGER' X,FS,I,J,K,L,Z,T;
     'REAL' X1,X2,B1,F1,A,MAX,MIN,NEW,NEWER;
     'BEGIN'
          'COMMENT' Both the max and min of the eigenvalue range are
          determined (MAX,MIN) as they are used as an initial guess
          depending on whether the smallest or largest eigenvalues
          are required;
          NEW←ABS(B[N])+ABS(D[N]); B[1]←0;
          MAX←C[N]+NEW;
          MIN←C[N]-NEW;
          'FØR' I←1 'STEP' 1 'UNTIL' N-1 'DØ'
          'BEGIN'
               NEW←ABS(B[I])+ABS(B[I+1]);
               'IF' P-1+I 'LE' N 'THEN'
               NEW←NEW+ABS(D[P-1+I]);
               'IF' I 'GE' P 'THEN' NEW←NEW+ABS(D[I]);
               NEWER←C[I]+NEW;
               NEW←C[I]-NEW;
               'IF' NEWER 'GT' MAX 'THEN' MAX←NEWER;
               'IF' NEW 'LT' MIN 'THEN' MIN←NEW;
          'END';
     'END';
     U[0]←S[0]←1;
     'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
     R[I,P]←D[I];
```

```
        'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
        V[I]←B[I];
        B1←0;
        X1←X2←MIN-1;
        L←1;
        Z←0;
        X←8;
        FS←P'/'2;
        'CØMMENT' The number of times that the Partial Sturm Sequence will
             be used before switching to the full sequence is preset to 8.
             The number of hands retained in the Partial sequence is preset
             to half the semi-bandwidth;
LAB1:   Z←Z+1
        'CØMMENT' The initial P-1 elements of the Sturm sequence can now
             be calculated, as they are the same for both the partial and
             full sequences;
        U[1]←C[1]-X2;
        'IF' U[1] 'EQ' 0 'THEN' U[1]←QZR;
        'FØR' I←2 'STEP' 1 'UNTIL' P-1 'DØ'
        'BEGIN'
             U[I]←C[I]-X2-V[I]*V[I]/U[I-1];
             'IF' U[I] 'EQ' 0 'THEN' U[I]←QZR;
        'END';
        'IF' Z 'LE' X 'THEN'
        'BEGIN'
             'CØMMENT' If less than X(8) iterations have been completed
                  then the Partial Sturm Sequence is used;
             'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
             'BEGIN'
                  A←R[I,P-1]←-R[I,P]*V[I-P+2]/U[I-P+1];
                  'FØR' K←P-2 'STEP' -1 'UNTIL' P-FS 'DØ'
                  'BEGIN'
                       T←I-K+1
                       A←-A*V[I]/U[T-1];
                       'IF' T 'GE' P 'THEN'
                       'FØR' J←FS+1-K 'STEP' -1 'UNTIL' 0 'DØ'
                       A←A-R[I,P-J]*R[T,P+1-K-J]/U[I-P+J+1];
                       R[I,K]←A;
                  'END';
                  A←B[I];
                  'IF' I-1 'GE' P 'THEN'
                  'FØR' J←FS-1 'STEP' -1 'UNTIL' 0 'DØ'
                  A←A-R[I,P-J]*R[I-1,P-1-J]/U[I-P+J+1];
                  V[I]←A;
                  A←C[I]-X2-A*A/U[I-1];
                  'FØR' J←FS 'STEP' -1 'UNTIL' 0 'DØ'
                  A←A-R[I,P-J]*R[I,P-J]/U[I-P+J+1];
                  'IF' A 'NE' 0 'THEN' U[I]←A'ELSE' U[I]←QZR;
             'END';
        'END'
        'ELSE'
        'BEGIN'
             'CØMMENT' After X(8) iterations the full Sturm sequence is used;
             'FØR' I←P 'STEP' 1 'UNTIL' N 'DØ'
             'BEGIN'
                  A←R[I,P-1]←-R[I,P]*V[I-P+2]/U[I-P+1];
```

```
                'FØR' K←P-2 'STEP' -1 'UNTIL' 3 'DØ'
                'BEGIN'
                     T←I-K+1;
                     A←-A*V[T]/U[T-1];
                     'IF' T 'GE' P 'THEN'
                     'FØR' J←P-K-2 'STEP' -1 'UNTIL' 0 'DØ'
                     A←A-R[I,P-J]*R[T,P+1-K-J]/U[I-P+J+1];
                     R[I,K]←A;
                'END';
                A←B[I]-A*V[I-1]/U[I-2];
                'IF' I-1 'GE' P 'THEN'
                'FØR' J←P-4 'STEP' -1 'UNTIL' 0 'DØ'
                A←A-R[I,P-J]*R[I-1,P-1-J]/U[I-P+J+1];
                V[I]←A;
                A←C[I]-X2-A*A/U[I-1];
                'FØR' J←P-3 'STEP' -1 'UNTIL' 0 'DØ'
                A←A-R[I,P-J]*R[I,P-J]/U[I-P+J+1];
                'IF' A 'NE' 0 'THEN' U[I]←A 'ELSE' U[I]←QZR;
           'END';
      'END';
      'CØMMENT' To avoid redetermining known eigenvalues the sequence
           is divided by these eigenvalues;
      'FØR' J←2 'STEP' 1 'UNTIL' L 'DØ'
      U[N+2-J]←U[N+2-J]/(EIG[J-1]-X2);
      'IF' B1 'EQ' 0 'THEN'
      'BEGIN'
           B1←1;
           X2←MIN;
           'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
           S[I]←U[I];
           'GØTØ' LAB1;
      'END';
      F1←1;
      'FØR' I←N 'STEP' -1 'UNTIL' 1 'DØ'
      F1←F1*S[I]/U[I];
      'IF' F1 'EQ' 1 'THEN' F1←F1+QZR;
      F1←(X2-X1)/(1-F1);
LAB3:
      'CØMMENT' A number of tests are made to ensure that convergence
           to an eigenvalue until: the correction factor (F1) is less
           than the required accuracy (EPS), more than 9 iterations
           have been performed;
      'IF' Z 'LT' 0 'THEN';
      'BEGIN'
           EIG[L]←X2-F1;
           S[N-L]←S[N-L]/('IF' EIG[L]-X1 'EQ' 0 'THEN' EPS 'ELSE' EIG[L]-X1);
           Z←0;
           L←L+1;
           'IF' L 'EQ' N1+1 'THEN' 'GØTØ' EXIT;
           MIN←EIG[L-1]+0.1;
           X1←X2←MIN-1;
           B1←0;
           'GØTØ' LAB1;
      'END';
      'IF' ABS(F1/(X2-F1)) 'LT' EPS 'AND' Z 'GT' X+1 'THEN' Z←-Z-2;
      'IF' ABS(F1/(X2-F1)) 'LT' EPS*0.1 'AND' ABS(Z) 'GT' X+2 'THEN'
```

```
          'BEGIN'
               Z←Z+2;
               'GØTØ' LAB3;
          'END';
     X1←X2;
     X2←X2-F1;
     'FØR' I←1 'STEP' 1 'UNTIL' N 'DØ'
     S[I]←U[I];
     'GØTØ' LAB1;
EXIT:
'END';
```

PROGRAM 22

This program is a modification to the Lanczos method for symmetric matrices as described in Chapter 7, and is programmed in Algol 68R. The procedure NORM is used, and simply normalises a given vector. Also the operators * and - have been defined for multiplication of two vectors, a vector and a matrix, a scalar and a vector, and subtraction of two vectors. These are simple routines and are not included.

```
'PRØC' ML=([,]'REAL'A,'INT'N,N1,N2,'REAL'EPS,QZR,'REF'[]'REAL'EIG):
'BEGIN'
     'C' A is the input matrix of order N. N1 is the number of the
     smallest eigenvalue required, N2 the largest. EPS is the accuracy
     required, and QZR the number closest to machine zero used to avoid
     division by small numbers. The eigenvalues are stored in the first
     N2-N1+1 elements of EIG. 'C';
     [1:N,1:N]'REAL' X;
     [1:N]'REAL' B,C,D,G;
     'REAL' MAX,MIN,NEW,NEWER,LAMBDA;
     'INT' S,L,K;
     'FØR' I 'TØ' N 'DØ'
     (X[I,1]←X[I,N]←0);
     X[1,1]←X[N,N]←1;
LAB1: D←A*X[,1];
     C[1]←X[,1]*D;
     B[1]←X[,N]*D;
     G[1]←(X[,1]*A*X[,N];
     X[,2]←D-(C[1]*X[,1])-(B[1]*X[,N]);
     NØRM(X[,2]);
     D←A*X[,2];
     B[2]←X[,1]*D;
     C[2]←X[,2]*D;
     'FØR' I 'FRØM' 3 'TO' N-1 'DØ'
     'BEGIN'
          G[I-1]←X[,N]*D;
          X[,I]←D-(C[I-1]*X[,I-1])-(B[I-1]*X[,I-2])-(G[I-1]*X[,N]);
          NORM(X[,I]);
          D←A*X[,I];
          B[I]←X[I-1]*D;
          C[I]←X[,I]*D
     'END';
     X[,N]←D-(C[N-1]*X[,N-1])-(B[N-1]*X[,N-2]);
     NØRM(X[,N]);
     D←A*X[,N];
     B[N]←X(,N-1]*D;
     C[N]←X[,N]*D;
     'C' The original matrix has now been transformed by the Lanczos
     method to one with main diagonal C, sub-diagonal B, with the
     remaining elements in the Nth row and column contained in G.
```

```
      The matrix is symmetric and has the same eigenvalues as A so
      bisection is now performed on this matrix. 'C';
      'BEGIN'
           'C' Limits are determined for the complete eigenvalue range
           (MAX,MIN). 'C';
           MAX←'ABS'(B[2])+'ABS'(B[1]);
           MIN←C[1]-MAX;
           MAX←C[1]+MAX;
           'FØR' I 'FRØM' 2 'TØ' N-2 'DØ'
           'BEGIN'
                NEW←'ABS'(B[I+1])+'ABS'(B[I])+'ABS'(G[I]);
                NEWER←C[I]+NEW;
                NEW←C[I]-NEW;
                (MAX<NEWER!MAX←NEWER);
                (MIN>NEW!MIN←NEW);
           'END';
           NEW←'ABS'(B[N])+'ABS'(B[N-1]);
           NEWER←C[N-1]+NEW;
           NEW←C[N-1]-NEW;
           (MAX<NEWER!MAX←NEWER);
           (MIN>NEW!MIN←NEW);
           NEW←'ABS'(B[1])+'ABS'(B[N]);
           'FØR' I 'FROM' 2 'TO' N-2 'DØ'
           NEW←NEW+'ABS'(G[I]);
           NEWER←C[N]+NEW;
           NEW←C[N]-NEW;
           (MAX<NEWER!MAX←NEWER);
           (MIN>NEW!MIN←NEW);
      'END';'REAL' ZER←0.1↑70;
      L←N1-1;
LABA: (L>=N2!'GØTØ' EXIT);
      'C' This is the major loop for the Lth eigenvalue.'C';
      LAMBDA←(MIN+MAX)*0.5;
LABB: 'BEGIN'
           'C' The Sturm sequence is now calculated, then bisection
           performed. 'C';
           'REAL' S,T,V;
           K←0;
           S←C[1]-LAMBDA;
           (S<0!K 'PLUS' 1);
           ('ABS'(S)<ZER!S←QZR);
           T←G[1];
           V←C[N]-LAMBDA;
           'FØR' I 'FROM' 2 TO' N-2 'DØ'
           'BEGIN'
                V←V-T*T/S;
                T←G[I]-T*B[I]/S;
                S←C[I]-LAMBDA-B[I]*B[I]/S;
                (S<0!K 'PLUS' 1);
                ('ABS'(S)<ZER! S←QZR);
           'END';
           V←V-T*T/S
           T←B[N]-T*B[N-1]/S;
           S←C[N-1]-LAMBDA-B[N-1]*B[N-1]/S;
           (S<0!K'PLUS' 1);
           ('ABS'(S)<ZER!S←QZR);
           S←V-T*T/S;
           (S<0!K 'PLUS' 1);
```

```
     'END';
     'IF' L-K >=0'THEN'
     'BEGIN'
          NEWER←MIN;
          MIN←LAMBDA;
          LAMBDA←LAMBDA+(LAMBDA-NEWER)*0.5
     'END'
     'ELSE'
     'BEGIN'
          LAMBDA←(LAMBDA+MIN)*0.5;
          'IF' (LAMBDA-MIN)*0.5/'ABS'LAMBDA<EPS
          'THEN'
          'BEGIN'
               MIN←2*LAMBDA-MIN;
               'FØR' I 'FROM' L+1 'TØ' (K>N2!N2!K) 'DØ'
               EIG[I-N1+1]←LAMBDA;
               L←K;
               'GØTØ' LABA
          'END' 'FI'
     'END''FI';
     'GØTØ' LABB
EXIT:
'END';
```