COMPUTING WITH LINEAR EQUATIONS AND MATRICES

R. F. Churchhouse
University College, Cardiff, United Kingdom

ABSTRACT

Systems of linear equations and matrices arise in many disciplines. The equations may accurately represent conditions satisfied by a system or, more likely, provide an approximation to a more complex system of non-linear or differential equations. The system may involve a few or many thousand unknowns and each individual equation may involve few or many of them. Over the past 50 years a vast literature on methods for solving systems of linear equations and the associated problems of finding the inverse or eigenvalues of a matrix has been produced. These lectures cover those methods which have been found to be most useful for dealing with such types of problem. References are given where appropriate and attention is drawn to the possibility of improved methods for use on vector and parallel processors.

1. INTRODUCTION

The topics to be covered in these few lectures have, over the past 50 years, been the subject of dozens of conferences, hundreds of books and thousands of research papers. A University course on the same topics would occupy at least 20 lectures. Since it is clearly impossible to go into great detail or generality it seems best to give a survey confined mainly to systems with real elements and not bothering with proofs or working out complicated algebraic expressions, of some of the methods which have been found most useful for solving a variety of problems involving systems of linear equations, matrix inversions and eigenvalues, pointing out the advantages and disadvantages of each and the type of problem where each particular method might be most appropriate. There is, of course, no single best method in most cases, claims that any particular method is best can usually be refuted by some suitably constructed counter-example.

The examples with which I shall illustrate the methods are, of necessity, based upon small scale systems in general, for the time and space required to read and write an n x n matrix are both proportional to n^2 and I see no value in spending an hour looking at the coefficients of a 30 x 30 matrix when a 3 x 3 will illustrate the point just as well, but real problems of course come in larger sizes, matrices of order 1000 or even 10000 being quite common, and there is at least one problem at CERN involving an asymmetric matrix of complex elements of order 10^5.

For anyone who wishes to know the details of any particular method or, say, its modification for complex elements appropriate references are given. For anyone interested in research let me say that although these subjects have been discussed at length for about 100 years there are not only fundamental mathematical problems still awaiting an answer but the

relatively recent introduction of vector and parallel processors calls for a re-examination of virtually all of the methods to be mentioned and opens up the possibility of new methods specifically designed to exploit these new machine architectures. In addition there is a class of problems of very large size where we would like to find a complete but approximate solution, and others where we would like a partial but accurate solution; in both cases a full and accurate solution would take too much time. Such problems, which arise in real-time data processing, such as signal analysis, high-energy physics and medicine, seem to be very difficult and there has been relatively little progress to date.

2.1 Some general observations

The most fundamental problem is that of solving a system of linear equations, if we can do this accurately and fast we are well equipped to tackle the related problems of matrix inversion and finding eigenvalues.

The basic idea behind most of the "direct" methods of solving systems of linear equations is to transform the system in such a way that the solution remains effectively the same but the equations are reduced to a simple form from which the solution is found easily. Transformations of such types can be achieved by "elimination" or by the use of "rotation" matrices.

If we can reduce a system of equations to diagonal or triangular form we can also reduce a matrix to such a form and hence we can invert a matrix.

If we wish to find the eigenvalues of a matrix we can transform the matrix so that the eigenvalues remain the same, by means of rotation matrices, but the matrix becomes much simpler, e.g. diagonal or tri-diagonal, and then use special techniques, if necessary, to obtain the eigenvalues of the transformed matrix.

Given any of these problems we will be dealing with a matrix of size, say, n x n which therefore contains n^2 elements. Before we can decide upon an appropriate method for solving the problem we need to answer certain basic questions

(1) Is n so large that storage of n^2 elements presents problems?
(2) Are most of the n^2 elements equal to zero (i.e. is the matrix 'sparse')?
(3) If the matrix is sparse are the non-zero elements distributed regularly within the matrix or in a random manner?
(4) Is it important to know the approximate run time required for the solution, i.e. which is preferable (i) the best solution achievable in a known fixed time or (ii) the most accurate solution that can be obtained regardless of time?
(5) Is a full solution required or is a partial solution sufficient? This particularly applies in eigenvalue problems.
(6) Is an estimate of the accuracy of the solution required?
(7) Is the problem one-off or one of a large number of similar problems?

(8) Is the problem to be run on a serial processor or on an array or vector processor?

2.2 Some standard notation and terminology

I	the unit matrix, of various sizes
A	an n x n real matrix
L	a lower triangular matrix
D	a diagonal matrix
U	an upper triangular matrix
$\underline{b}$	a column vector with n known elements
$\underline{x}$	a column vector with n unknown elements
λ	an eigenvalue
A^T	the transpose of A

Symmetric matrix — a matrix A having the property that $A^T = A$.

Orthogonal matrix — a matrix A such that $A^TA = I$.

Positive Definite Matrix — A is a positive definite matrix if it has the property that for all real vectors $(\underline{x}) \neq (\underline{0})$, $\underline{x}'A\underline{x} > 0$.

Rotation matrix — a matrix R(i,j) which is identical to I except for the four elements with both suffixes equal to i or j when $r_{ii} = r_{jj} = \cos(\theta)$, $r_{ij} = -r_{ji} = \sin(\theta)$ for some angle θ.

Inner product of two real vectors $\underline{x} = (x_1, x_2, \ldots, x_n)$ and $\underline{y} = (y_1, y_2, \ldots y_n)$ is denoted by (x,y) and defined to be

$$(\underline{x},\underline{y}) = \sum_{i=1}^{n} x_i y_i$$

Infinity (or Chebyshev) norm of a vector $\underline{x}$

$$||\underline{x}||_\infty = \max_{i=1,..,n} |x_i|$$

(i.e. $||\underline{x}||_\infty$ is the absolutely largest element of $\underline{x}$).

Ill-conditioned matrix — In problems such as we are considering the concepts of well-conditioned and ill-conditioned matrices naturally arise. A matrix is said to be ill-conditioned in the context of a particular problem if small changes in the elements of the matrix produce large changes in the elements of the solution to that problem. A matrix which is ill-conditioned in the context of

inversion will not necessarily be ill-conditioned so far as finding its eigenvalues are concerned. It is not generally possible to say at a glance that a matrix is ill conditioned in some context but a variety of "condition numbers" have been proposed which indicate how ill-conditioned (or otherwise) a matrix might be. (See [7], 88-91).

Example 1 An ill conditioned matrix and its inverse.
The Hilbert Matrix of order n, H_n, is defined by

$$H_n = (a_{ij}) = \frac{1}{(i+j-1)} \qquad i,j = 1,2,\ldots n$$

This matrix occurs naturally in the problem of Least Squares polynomial approximation. It can be shown that the (i,j)-th element (α_{ij}) in the inverse H_n^{-1} is

$$\alpha_{ij} = \frac{(-1)^{i+j}(n+i-1)!(n+j-1)!}{(i+j-1)\,[\,(i-1)!(j-1)!\,]^2(n-i)!(n-j)!}$$

If we put n=(2k+1),i=j=k in this expression and use Stirling's approximation for k! we see that

$$\alpha_{kk} \text{ is of order } (3^{6k})$$

i.e. grows exponentially with k, even though the smallest element in H_n is $(4k+1)^{-1}$ and the largest is 1. As an example, when n = 3

$$H_3 = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}, \quad H_3^{-1} = \begin{pmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{pmatrix}$$

2.3 Computer arithmetic

Arithmetic, as performed on computers, is not the same as the (ideal) arithmetic of Pure Mathematics, nor is it even the same as the arithmetic of everyday life. Anyone who uses a computer to carry out numerical calculations will know about rounding errors but there is no known method for estimating the likely magnitude of the accumulated rounding error following an extensive sequence of numerical operations. Where bounds are available at all they are usually hopelessly pessimistic. As to the accuracy of the solution: it is by no means a trivial matter to ensure that all the elements of a solution are correct to, say, k decimal places; not only are the machine word-size and arithmetic relevant but so also are the convergence and stability properties of the method used and the ill-conditioning, or otherwise, of the data. It is even possible that the precise order in which the arithmetic operations are carried out will affect the result since, for

example, computer arithmetic is not always associative, i.e. it is not necessarily true that

$$(x+y) + z = x + (y+z)$$

and so two mathematically identical methods may produce different results - for further details and examples see [1] and [2].

So far as these lectures are concerned truncation error, which is so important in most branches of numerical analysis, is not relevant.

3. SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

Suppose we have a system of m equations in n unknowns:

$$\begin{aligned} a_{1,1}\,x_1 + a_{1,2}\,x_2 + \ldots + a_{1,n}\,x_n &= b_1 \\ a_{2,1}\,x_1 + a_{2,2}\,x_2 + \ldots + a_{2,n}\,x_n &= b_2 \\ \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \qquad\quad &\vdots \\ a_{m,1}\,x_1 + a_{m,2}\,x_2 + \ldots + a_{m,n}\,x_n &= b_m \end{aligned} \tag{3.1}$$

then the following well-known theorem holds:

Theorem (i) If m=n and $(\underline{b}) \neq (\underline{0})$ the system (3.1) has a unique solution if and only if the determinant, det A, of the matrix A is not equal to zero.

(ii) If m<n the system will, in general, have an infinity of solutions.

(iii) If m>n the system will, in general, have no solution.

We shall only be concerned with the case m=n but it is worth remarking that the case m>n is frequently of interest when, although there is no solution vector $(\underline{x})$ satisfying all the equations exactly, we try to find that vector $(\underline{x}^1)$ which, in some sense, 'best satisfies' the equations; for a discussion of such topics see [3].

Methods for the solution of systems of linear equations are generally classified either as direct methods, or as indirect methods, though there are methods, such as the Conjugate Gradient method (which we shall not discuss, for details see [4, pp442-445]) which might be considered as hybrid in that they are partly direct and partly indirect. Direct methods, if we ignore rounding errors, will in theory lead to the exact solution, and will in practice lead to an approximate solution, in a finite number of steps which can be predicted in advance. Indirect methods on the other hand, begin with an approximate solution (which may be entirely arbitrary, such as $(\underline{x}) = (1,1,..,1)$) and attempt to 'improve' upon this at each stage (iteration); the process may or may not converge and if it does converge the number of iterations required to obtain a solution of given accuracy cannot (in general) be specified in advance. It might appear from these remarks that all the advantages lie with direct methods, but this is not so, as we shall see.

3.1 Direct Methods

In theory the problem of solving a system such as (3.1) (with m=n) is trivial for we can write down a closed formula

containing a finite number of terms which gives the solution. For let A_i denote the matrix obtained by replacing the i-th column of A by the vector $\underline{b}$ then

$$x_i = \det(A_i)/\det(A) \tag{3.2}$$

This is Cramer's Rule, which is valid if A is not singular, i.e. if $\det(A) \neq 0$. Unfortunately the direct evaluation of these (n+1) determinants involves more than (n+1)! multiplications and divisions and is therefore totally impracticable on any existing computer for n as small as, say, 15 and is not to be recommended even for n=5.

3.1.1 Gaussian Elimination The essentials of this method are faimilar to anyone who has had algebra lessons at school but the finer details are still a matter of controversy. Essentially: starting from the system

$$\begin{aligned} a_{1,1}\,x_1 + a_{1,2}\,x_2 + \ldots + a_{1,n}\,x_n &= b_1 \\ a_{2,1}\,x_1 + a_{2,2}\,x_2 + \ldots + a_{2,n}\,x_n &= b_2 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\quad & \vdots \\ a_{n,1}\,x_1 + a_{n,2}\,x_2 + \ldots + a_{n,n}\,x_n &= b_n \end{aligned} \tag{3.3}$$

and, assuming that $a_{1,1} \neq 0$, we subtract $a_{i,1}/a_{1,1}$ times the first equation from the i-th equation for i=2,3,...n. This leads to a modified system

$$\begin{aligned} a_{1,1}\,x_1 + a_{1,2}\,x_2 + \ldots + a_{i,n}\,x_n &= b_1 \\ a^{(1)}_{2,2}\,x_2 + \ldots + a^{(1)}_{2,n}\,x_n &= b^{(1)}_2 \\ a^{(1)}_{3,2}\,x_2 + \ldots + a^{(1)}_{3,n}\,x_n &= b^{(1)}_3 \\ \vdots \qquad\qquad \vdots \qquad\quad & \vdots \\ a^{(1)}_{n,2}\,x_2 + \ldots + a^{(1)}_{n,n}\,x_n &= b^{(1)}_n \end{aligned} \tag{3.4}$$

in which x_1 appears only in the first equation. Assuming that $a^{(1)}_{2,2} \neq 0$ we now subtract $a^{(1)}_{i,2}/a^{(1)}_{2,2}$ times the second equation from the i-th in (3.4) for i=3,4,..n, which leads to a new modified system in which only the first equation contains x_1 and only the first and second equations contain x_2.

This elimination process is continued for (n-3) more stages at the end of which we are left with a system of equations whose matrix is upper-triangular

$$
\begin{aligned}
a_{1,1}\,x_1 + a_{1,2}\,x_2 + \ldots + a_{1,n}\,x_n &= b_1 \\
a^{(1)}_{2,2}\,x_2 + \ldots + a^{(1)}_{2,n}\,x_n &= b^{(1)}_2 \\
\ldots + a^{(2)}_{3,n}\,x_n &= b^{(2)}_3 \\
\vdots \qquad\qquad & \quad \vdots \\
a^{(n+1)}_{n,n}\,x_n &= b^{(n-1)}_n
\end{aligned}
\tag{3.5}
$$

From the last equation of (3.5) x_n is found; the value of x_n is then substituted in the penultimate equation and so the value of x_{n-1} is found; this process of back-substitution is continued (n-2) more times so that x_{n-2}, x_{n-3}, .., x_2, x_1 are found in succession, and the solution is complete. It can be shown that the number of multiplications required is about $\frac{1}{3}n^3$ which is an enormous improvement on (n+1)! unless n is very small.

So much for the essentials of Gaussian elimination, now for the refinements. In the first place we assumed that $a_{1,1} \neq 0$, then that $a^{(1)}_{2,2} \neq 0$, then that $a^{(2)}_{3,3} \neq 0$, and so on, until the elimination was complete when we assumed that $a^{(n-1)}_{n,n} \neq 0$. If any of these coefficients in the pivotal positions (or "pivots" as they are called) is zero the method, as described, breaks down. Recovery is, however, easy enough; suppose $a^{(k-1)}_{k,k} = 0$, find an element in the same column $a^{(k-1)}_{r,k}$ (r>k) which is not equal to zero; interchange the k-th and r-th rows (equations) and proceed as before. If no such element $a^{(k-1)}_{r,k}$ exists the matrix is singular.

Some interchange process such as the one just described must be carried out if a pivot turns out to be zero but there is also a good case for taking similar action when the pivot, although not zero, is small in absolute value compared to other elements in the matrix. The smaller the pivot is the more significant, relatively, will be its rounding error and the larger and more inaccurate will be its reciprocal which we will be using in the elimination process and so the more inaccurate will be the entire solution. In view of this it is advisable not to use "small" pivots - everyone agrees on this - but there are different views on how acceptable pivots should be chosen. There are three standard methods:

(i) Find the (absolutely) largest element in the same column as the pivot and below it and interchange rows to bring this element to the pivotal position. Thus if a $a^{(k-1)}_{k,k}$ is small and if

$$
a^{(k-1)}_{r,k} = \operatorname*{Max}_{k<i\leqslant n} \left| a^{(k-1)}_{i,k} \right|
$$

then interchange rows r and k including b_r and b_k. This process is known as partial pivoting.

(ii) Find the (absolutely) largest element in the submatrix with a $a_{k,k}^{(k-1)}$ as its top left corner; if this element is a $a_{i,j}^{(k-1)}$ interchange rows k and i (including b_k and b_i) and also interchange columns k and j in the complete matrix.
This last step is essential for we have interchanged the variables x_k and x_j when we interchanged the columns.

This process is known as complete pivoting.

(iii) Find the first element, in the same column as the pivot if possible but anywhere in the submatrix if necessary, which exceeds (in absolute value) some given threshold; make row and column interchanges as necessary. The threshold may be fixed or may vary from row to row. There doesn't appear to be a recognised phrase for this type of pivoting; I usually refer to it as "threshold pivoting".

Obviously the use of any form of pivoting increases the number of operations (and hence the time) required for the solution; the reward should be that rounding errors are reduced. There are, however, different opinions as to which method of pivoting is best. Wilkinson [5], on the whole favours complete pivoting on the grounds that it is never a bad method and that no consistently better one has been proposed. A method proposed about 30 years ago which is none of the above is to choose as pivot that element closest to a specific value - the n-th root of the absolute value of the determinant of A - but Wilkinson [6] gives arguments against this approach (one being that the value of the determinant is in general not known until the elimination process has been completed).

3.1.2 Modes of pivoting: some experimental evidence

Since theoretical comparisons of the accuracy achievable by the various modes of pivoting are not entirely satisfactory, because of the random nature of accumulation of rounding errors an experiment was carried out on the IBM computer at CERN. A large number of systems of equations in up to 20 variables was generated as follows: for a system of n variables n^2 (pseudo-) random variables uniformly distributed over the interval <-1,1> were taken as the coefficients of the nxn matrix A; the right hand side vector $\underline{b}$ was then generated by forcing the solution vector $\underline{x}$ to be of a particular type e.g. $\underline{x} = (1,2,3,..n)$. By fixing the solution in this way an exact measure of the errors in each computed solution was available and so comparison of errors in solutions obtained by different methods was possible. The measure used was L_1, so that if the computed solution is $\underline{x} = (\hat{x}_1, \hat{x}_2, .., \hat{x}_n)$ and the exact solution is $\underline{x} = (x_1, x_2, .. x_n)$ then the

error measure associated with that solution is

$$e = \sum_{j=1}^{n} |x_j - \hat{x}_j| \qquad (3.6)$$

Other measures, such as L_∞ or L_2 could have been used; the results would probably not have been very different. If the same system of equations is solved by two different methods and errors e_1 and e_2 are obtained the ratio e_1/e_2 can be used to indicate the superiority (if the ratio is less than 1) or inferiority (ratio > 1) of the first method compared to the second. In the program four pivoting modes were compared (i) no pivoting at all, (ii) partial pivoting, (iii) complete pivoting, (iv) threshold pivoting, with a variety of thresholds; all modes were tested on several thousand cases. The overall conclusions were quite clear:

(i) Except in the case of very small matrices ($n \lesssim 3$) any form of pivoting will usually lead to smaller error, as measured by (3.6), compared to the solution obtained without pivoting.

(ii) The improvement obtained by pivoting increases as n, the matrix size, increases; (e.g. at n=10 all pivoting methods cut the error measure by a factor of about 2 on average and at n=20 by a factor of about 5).

(iii) Complete pivoting did not produce results (for matrices up to size 20) which were substantially better than partial pivoting.

(iv) Threshold pivoting with the threshold set at about half the expected maximum element is about as good as partial or complete pivoting as regards accuracy.

(v) Both partial pivoting and threshold pivoting save substantially on the number of comparisons compared to complete pivoting - complete pivoting approximately doubles the time required to solve the equations.

One additional feature was noticed in the results for partial pivoting: if the equations were ordered in such a way that

$$|x_n| \lesssim |x_{n-1}| \lesssim \cdots \lesssim |x_1|$$

(so that the absolutely smallest variable is found first, etc.) the accuracy of the solution was usually improved. In general of course the relative sizes of the variables are not known, so the observation will rarely be usable.

It must be stressed that all of these observations apply only "on average"; the programs recorded a variety of extreme cases e.g. a 10x10 matrix where the use of complete pivoting produced an error measure 21 times worse than when no pivoting at all was used! The example below is a well-known extreme case.

Example 2 A case where partial pivoting produces the worst possible elements.

$$\text{Let } A = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots. & 0 & 1 \\ -1 & 1 & 0 & 0 & \ldots. & 0 & 1 \\ -1 & -1 & 1 & 0 & \ldots. & 0 & 1 \\ . & . & & & & . & . \\ . & . & & & & . & . \\ . & . & & & & . & . \\ -1 & -1 & & \ldots. & & \ldots.-1 & 1 \end{pmatrix}$$

Using the diagonal element as pivot (as the largest element in its column) we get the system

$$\begin{pmatrix} 1 & 0 & 0 & \ldots. & 0 & 1 \\ 0 & 1 & 0 & \ldots. & 0 & 2 \\ 0 & 0 & 1 & \ldots. & 0 & 4 \\ & & & & & \\ & & & & & \\ 0 & 0 & & \ldots\ldots\ldots & & 2^{n-1} \end{pmatrix}$$

- which gives $a_{nn}=2^{n-1}$, the theoretically worst possible growth.

[For <u>complete</u> pivoting Wilkinson has obtained the bound:

$$\text{Max}|a_{ij}| < \left(N.2^{\frac{1}{1}}.\ 3^{\frac{1}{2}}\ 4^{\frac{1}{3}}\ \ldots\ N^{\frac{1}{N-1}}\right)^{\frac{1}{2}} < 2N^{\frac{1}{2}}.\ N^{\frac{1}{4}\log N}$$

- but this, though much smaller than 2^{N-1} is extremely pessimistic and no no matrices are known which approach it.]

One further point concerning pivoting. If the matrix is sparse we may be reluctant to use pivoting, particularly complete pivoting, since it may destroy a regular pattern of non-zero terms or produce a lot of "fill-in". If pivoting <u>is</u> used in sparse cases a criterion for choosing the pivot due to Markowitz (see [17], p.98-107) seems to be satisfactory: find a non-zero element that is bigger than some threshold and which has the smallest product of the numbers of non-zero elements in its row and in its column and make this the pivot.

Gaussian elimination has been covered in some detail because it is the most fundamental and most widely-used direct method and is easy to program under any of the pivoting modes described above.

3.1.2. <u>The Gauss-Jordan method</u>. This is a variant of the Gaussian elimination method in which at the k-th stage of the elimination the k-th equation (after pivoting) is used to eliminate x_k from <u>all</u> the equations except the k-th. At the end of the elimination process we are left with a purely diagonal matrix which leads to the solution at once, without the need for back-substitution. The method is in practice rarely used for solving linear equations since it requires approximately $\frac{1}{2}n^3$ multiplications, 50% more than Gaussian elimination; for matrix inversion however, if used appropriately, it is more economical on storage and takes exactly the same number of operations as the Gaussian elimination method (see Section 5).

Having obtained a solution it would be nice if we could decide easily how accurate it was. The "obvious" test is to substitute the values obtained for the variables into the original equations and see how well the computed r.h.s. vector agrees with the value of the r.h.s. vector $\underline{b}$. If the equations are well-conditioned this is a reasonable test but when the equations are ill-conditioned a "residual vector" with very small components does not necessarily indicate an accurate solution. There are numerous examples in the literature of very small component residual vectors where the solution is very badly wrong. For another such illustration see Example 3, which also illustrates that pivoting doesn't always lead to a better solution.

Example 3 to illustrate that (i) pivoting doesn't always lead to a more accurate solution and (ii) that small residuals do not necessarily indicate a very accurate solution.

If we solve the equations

$$0.343x + 0.49y + 0.7z = 3.423$$
$$0.512x + 0.64y + 0.8z = 4.192$$
$$0.729x + 0.81y + 0.9z = 5.049$$

working throughout to 4d.p., without pivoting we obtain the triangular array

$$0.3430x + 0.4900y + 0.7000z = 3.4230$$
$$0.0914y + 0.2449z = 0.9175$$
$$0.0322z = 0.0966$$

which gives z=3,y=2,x=1, the exact solution. If we use partial pivoting we obtain the triangular array

$$0.7290x + 0.8100y + 0.9000z = 5.0490$$
$$0.0711y + 0.1679z = 0.6461$$
$$0.0193z = 0.0578$$

which gives z=2.9948,y=2.0155,x=0.9892 and we see that the errors are quite large. Nevertheless, if we substitute these values in the original equations the r.h.s. values are found to be in error only by (-0.0003, -0.0002,0) to 4 d.p. which might be taken as indicating an accurate solution.

The system is nearly singular, the determinant being about 0.0005, but this doesn't explain why pivoting produces a poor result whereas no pivoting produces an accurate one.

3.1.3 Triangularization If, in the Gaussian elimination process we denote the multiplier $a_{ij}^{(j-1)}/a_{jj}^{(j-1)}$, which is used to eliminate x_j from the i-th equation, by m_{ij} then, if no row or column interchanges have taken place, it is easy to verify that the product of the lower triangular matrix, L (say)

$$L = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ m_{12} & 1 & 0 & \dots & 0 \\ m_{13} & m_{23} & 1 & \dots & 0 \\ \vdots & \vdots & & & \vdots \\ m_{in} & m_{2n} & & \dots & 1 \end{pmatrix} \tag{3.7}$$

and the upper triangular matrix U (say) given by (3.5) is (to within rounding errors) our original matrix A; thus we have factorised A into the product of a lower triangular matrix and an upper triangular matrix

$$A = LU \tag{3.8}$$

Furthermore since

$$\det(A) = \det(L)\,\det(U)$$

and $\det(L) = 1$ so

$$\det(A) = \det(U) = a_{11}a_{22}^{(1)}\,a_{33}^{(2)}\,\dots a_{nn}^{(n-1)} \tag{3.9}$$

and we see that the Gauss process enables us to find det(A) in $O(n^3)$ operations as a by-product.

If row or column interchanges have taken place a factorisation analogous to (3.8) is still obtained but the matrices A and L must be modified by appropriate row or column permutations at each stage.

3.1.4 Solving for several right-hand sides If we wish to solve $A\underline{x} = \underline{b}$ for several column vectors $\underline{b}_1, \underline{b}_2, \dots, \underline{b}_r$ which are known in advance the elimination process can be applied to the elements of the nxr matrix $(\underline{b}_1, \underline{b}_2, \dots, \underline{b}_r)$ in parallel. If the vectors $\underline{b}_i$ are not known in advance then the matrices L, U of (3.8) should be retained and, as the various vectors $\underline{b}_i$ become available the equations $A\underline{x} = \underline{b}_i$ are solved by treating them as $LU\underline{x} = \underline{b}_i$ and then solving firstly $L\underline{y} = \underline{b}_i$ and then $U\underline{x} = \underline{y}$, both of which involve only back-substitution and require only $\frac{1}{2}n^2$ operations.
Alternatively we could compute and store A^{-1}, using the fact that $A^{-1} = U^{-1}L^{-1}$ and since L and U are triangular their inverses are easily found; the solution to $A\underline{x} = \underline{b}_i$ is then, of course, $\underline{x} = A^{-1}\underline{b}_i = U^{-1}L^{-1}\underline{b}_i$, but this also requires n^2 multiplications so there is no saving in time by using this method.

3.1.5 Compact methods The factorization A = LU given by (3.8) is just one of an infinity of such factorisations, for if we write down L and U as general lower- and upper-triangular matrices they each contain $\frac{1}{2}n(n+1)$ unknown elements, making n^2+n unknowns in all and equating the elements of their product with the elements of A produces only n^2 equations; therefore we can hope to give n of the unknowns any values we wish and thus obtain

unique L and U. In the case of (3.8) the diagonal elements of L are all equal to 1; this is the natural form resulting from Gaussian elimination. When obtained in this way the method based upon it is usually known as Doolittle's method. If on the other hand we decide to take the diagonal elements of U all equal to 1 the method is known as Crout's method. Methods based upon such factorisations are known as "compact" methods; their computational advantage is that the elements of the matrices L, U can be obtained with less round-off error than in the elimination method; if double-precision is used for the calculation of these elements and the values finally rounded to single-precision the solutions of the associated equations should be more accurate than when obtained by elimination.

In the special case when A is symmetric and positive-definite, such as happens in many applications, we can take $U = L^T$ which enables us to save both space and time, since only about $\frac{1}{6}n^3$ operations are required to complete the factorisation (which is known as Choleski factorisation). If A is symmetric but not positive definite some diagonal elements of U may be complex.

For details of all these, and other, methods see [7, p10-16].

3.2 Scaling

It is generally agreed that it is numerically advantageous if all the elements, a_{ij}, of the coefficient matrix are of approximately the same order of magnitude. In general it may not be easy to arrange this but one simple method which leads part-way to this objective is row scaling, in which each equation is multiplied by a constant which is chosen to make its largest element, in absolute value, equal to 1. Row scaling does not affect the solution but if column scaling is used so that, for example, the k-th column is multiplied by c_k then we need to remember that the computed variable x_k must be replaced by $c_k x_k$ for the solution.

3.3 Indirect methods

The basic idea underlying all indirect methods for systems of linear equations is to re-write the equations in an iterative form, take some approximation to the solution vector and iterate until the solution vector is stationary. The idea can be illustrated very nicely by a simple example. Suppose we wish to solve

$$\begin{aligned} 2x + y &= 3 \\ x - 2y &= 4 \end{aligned} \tag{3.10}$$

We convert the equations to the iterative form

$$\begin{aligned} x_{n+1} &= \tfrac{1}{2}(3-y_n) \\ y_{n+1} &= \tfrac{1}{2}(x_{n+1}-4) \end{aligned} \tag{3.11}$$

and take an arbitrary starting vector (x_o, y_o) as our approximation to the

solution; (0,0) will do. Using the first equation in (3.11) we obtain x_1=1.5, and then putting x_1=1.5 in the second equation of (3.11) we obtain y_1=-1.25; using y_1 in the first equation, then gives x_2=2.125 and so on; eventually we find x_7=2.000, y_7=-1.000, both of which are correct to 3.d.p. The process so illustrated is the Gauss-Seidel process. In the example above the process converged to the solution; it will not always do so as can easily be seen by taking the same example and writing the equations in reverse order leading to the iterative form

$$x_{n+1} = 4+2y_n$$

$$y_{n+1} = 3-2x_{n+1} \qquad (3.12)$$

We again start from (0,0) and obtain $x_1 = 4$, $y_1 = -5$, $x_2 = -6$, $y_2 = 15$, $x_3 = 34$, $y_3 = -65$ etc. and it is clear that the iteration is diverging rapidly. It is easy to prove in fact that if the Gauss-Seidel process is applied to the equations

$$ax + by = m_1$$

$$cx + dy = m_2$$

then convergence will occur if and only if $|ad|>|bc|$, and this in turn implies that one of the two orderings of the equations will lead to convergence and the other to divergence unless $|ad|=|bc|$ when the process cycles indefinitely.

Analysis of the Gauss-Seidel method in the case of two equations is easy and complete; unfortunately in the case of n equations no such simple analysis is possible. For the n equations given by (3.3) the Gauss-Seidel iterative form is (assuming none of the diagonal elements a_{kk} is zero)

$$\begin{aligned}
x_1(k+1) &= \frac{1}{a_{11}} \quad (b_1-a_{1,2}x_2^{(k)}-a_{1,3}x_3^{(k)}-\ldots-a_{1,n}x_n^{(k)}) \\
x_2(k+1) &= \frac{1}{a_{22}} \quad (b_2-a_{2,1}x_1^{(k+1)}-a_{2,3}x_3^{(k)}-\ldots-a_{2,n}x_n^{(k)}) \\
&\vdots \\
x_n^{(k+1)} &= \frac{1}{a_{nn}} \quad (b_n-a_{n,1}x_1^{(k+1)}-a_{n,2}x_2^{(k+1)}-\ldots a_{n,n-1}x_{n-1}^{(k+1)})
\end{aligned} \qquad (3.13)$$

Note the important point that in the r.h.s. of the j-th equation in (3.13) the variables $x_1, x_2 \ldots, x_{j-1}$ occur as $x_1^{(k+1)}$, $x_2^{(k+1)}$, ..., $x_{j-1}^{(k+1)}$ whereas the variables x_{j+1}, $x_{j+2}, \ldots x_n$ occur as $x_{j+1}^{(k)}$, $x_{j+2}^{(k)}$, .., $x_n^{(k)}$ - in other words we use the latest available values of all variables. In the Jacobi method, by contrast, in computing $x_j^{(k+1)}$ we use the k-th iterates of all the other

variables $x_1^{(k)}$, $x_2^{(k)}$,.., $x_n^{(k)}$.

The Gauss-Seidel and Jacobi methods are two examples of a class of iterative methods, as we now show. Write A in the form

$$A = L + D + U$$

and suppose that D, a diagonal matrix, has no zero elements on its diagonal (if A is non-singular this can always be arranged, by permuting rows and columns as necessary). The equations $A\underline{x} = \underline{b}$ can be written

$$D\underline{x} = -(L+U)\underline{x}+b \tag{3.14}$$

and this suggests the iterative form

$$D\underline{x}^{(k+1)} = -(L+U)\underline{x}^{(k)}+\underline{b} \tag{3.15}$$

which is the Jacobi iterative method.

Alternatively we could write the equation as

$$(L + D)\underline{x} = -U\underline{x}+\underline{b} \tag{3.16}$$

leading to the iterative form

$$(L + D)\underline{x}^{(k+1)} = -U\underline{x}^{(k)}+\underline{b} \tag{3.17}$$

which is the Gauss-Seidel iterative method.

The solutions to (3.14) and (3.16) can be written down formally as

$$\underline{x} = -D^{-1}(L+U)\underline{x}+D^{-1}\underline{b} \tag{3.18}$$

and

$$\underline{x} = -(L+D)^{-1}U\underline{x}+(L+D)^{-1}\underline{b} \tag{3.19}$$

and these lead to the conditions for convergence of the two methods, thus, for example, the Gauss-Seidel method converges (independently of the initial approximation) if and only if all eigenvalues of the matrix $(L+D)^{-1}U$ have modulus less than one (for a proof see [10,p102-139]).

A generalisation of (3.14) and (3.16) can be obtained by splitting A into the sum of two matrices B and C, say, then

$$A\underline{x} = \underline{b}$$

is equivalent to $(B+C)\underline{x} = \underline{b}$

or $B\underline{x} = -C\underline{x}+\underline{b}$

which suggest the iterative form

$$B\underline{x}^{(k+1)} = -C\underline{x}^{(k)}+b \tag{3.20}$$

and for the condition for convergence of (3.20) is that the eigenvalues of the matrix $B^{-1}C$ all have modulus less than unity.

A particularly important iterative method is SOR (successive over-relaxation) which is widely used in the solution of systems of linear equations arising from finite difference schemes for solving elliptic P.D.E's. In this method the value taken for $x_1^{(k+1)}$ is

$$x_i^{(k+1)} = \omega Z_i^{(k+1)} + (1-\omega)x_i^{(k)} \qquad (3.21)$$

where $Z_i^{(k+1)}$ is the expression, given by (3.13), for $x_i^{(k+1)}$ i.e. the normal Gauss-Seidel. It is usual to choose values of ω in the interval (0,2) since this ensures convergence in certain important cases; an optimal choice of ω can dramatically speed up convergence but the actual choice is often a matter of trial and error. When $\omega=1$, as is obvious from (3.21) the method is identical to the Gauss-Seidel. For detailed analysis of such systems see [10].

The convergence criteria quoted above are mathematically satisfying but in practice we will not know whether the eigenvalues of $B^{-1}C$ all lie within the unit circle or not. In the case of the Gauss-Seidel and Jacobi methods sufficiency conditions for convergence are available which apply to many of the systems of equations encountered in engineering and allied problems; the most useful single sufficiency condition is given by:

Theorem 3.1 If the coefficients a_{ij} of A possess the property that

$$2|a_{ii}| > \sum_{j=1}^{n} |a_{ij}|$$

then the Gauss-Seidel method applied to the system $A\underline{x}=\underline{b}$ will converge independently of the starting vector. (For a proof, and further discussion, see [11,pp.61-81].)

Another criterion which is often quoted is that the G-S method will converge if A is positive definite. This may not be very useful in general but there is one case where it can be exploited if we are really desperate (though I don't recommend it, unless all else fails): if A is real then A^TA is symmetric and positive definite and so if we take the system $A\underline{x}=\underline{b}$ and premultiply both sides by A^T we obtain the equivalent system $A^TA\underline{x} = A^T\underline{b}$ and since A^TA is positive definite the G-S method applied to this system will converge - but, be warned, the convergence is likely to be slow. The following illustrates the technique.

Example 4 (Forcing the Gauss-Seidel iteration to converge)

If we apply the Gauss-Seidel method to the system

$$x + 2y + z = 0$$

$$2x + y + z = 2$$

$$x + y + z = 1$$

starting from (0,0,0) we obtain (0,2,-1) as the first iterate and (-3,9,-5) as the second. Since the solution is (1,-1,1) it is clear that we have divergence. If however we multiply the matrix and r.h.s. by the transpose coefficient matrix we obtain an equivalent system with a positive definite coefficient matrix viz:

$$6x + 5y + 4z = 5$$
$$5x + 6y + 4z = 3$$
$$4x + 4y + 3z = 3$$

and the Gauss-Seidel process will now converge even though the matrix is far from diagonally dominant. Starting from (0,0,0) as before we obtain, to 3 d.p.:

(i) after 5 iterations (1.007,-0.629,0.496)

(ii) after 10 iterations (1.041,-0.732,0.588)

and convergence is occurring, though slowly.

As to the speed of convergence of these various methods it can be proved that the Jacobi iterative method is inferior to Gauss-Seidel since the latter converges whenever the former does and furthermore converges twice as fast (so that the G-S requires only half as many iterations as the Jacobi to achieve the same accuracy).

It should be pointed out however that on a parallel processor it is quite possible that the Jacobi method might converge to a solution more quickly than the Gauss-Seidel method since all the n values at the end of the last iteration are available and so all new values can be calculated simultaneously whereas in the Guass-Seidel computation of $x_i^{(k+1)}$ cannot be started until $x_{i-1}^{(k+1)}$ has been found. There might be scope here for some hybrid method being used, and the topic is worthy of further research.

3.3.1. Accelerating the convergence of iterative methods The speed of convergence of the iterative method (3.20) is determined by the spectral radius (= magnitude of the dominant eigenvalue) of the matrix $B^{-1}C$, $|\lambda_1|$, say. The smaller $|\lambda_1|$ is the faster the convergence. There is a considerable literature [10] on how we might modify iterative methods in order to ensure that $|\lambda_1|$ is small; unfortunately in most cases a certain amount of trial-and-error is necessary. There is however a very simple, general, technique for accelerating the convergence of any linearly converging iteration, such as we have in the Gauss-Seidel and similar processes. For if $||E_{n+1}||$ denotes a norm of the error vector, E_{n+1}, after (n+1) iterations then

$$||E_{n+1}|| \doteq |\lambda_1| \, ||E_n|| \tag{3.22}$$

and therefore, since (3.22) is linear, Aitken's δ^2 Extrapolation process can be applied [12, p.46-47; 72-73] viz: if $x_i^{(n-1)}$, $x_i^{(n)}$ and $x_i^{(n+1)}$ are three consecutive approximations to the i-th element x_i of the solution vector $\underline{x}$ obtained by an iterative method then

$$\hat{x}_i = x_i^{(n+1)} - \frac{(x_i^{(n+1)} - x_i^{(n)})^2}{x_i^{(n+1)} - 2x_i^{(n)} + x_i^{(n-1)}} \tag{3.23}$$

will, in general, be a better approximation. Note however that the expression on the r.h.s. of (3.23) will eventually involve very small numbers in both the numerator and denominator so the formula should be used judiciously; its main use is in accelerating convergence at a relatively early stage.

4. OPERATIONS WITH MATRICES

4.1. Sparse matrices

Many problems which occur in engineering and other disciplines give rise to large systems of equations where each equation involves very few unknowns. The coefficient matrix in such a case consists mainly of zeros; typically, each of the n equations (say) involves at most k unknowns, k being a small constant. The number of non-zero coefficients is therefore at most kn. An nxn matrix containing 'few' non-zero elements is called sparse; 'few' is generally taken to mean that the total number of non-zero elements is bounded by a constant multiple of n. If the few non-zero elements in each row are in adjacent columns, which frequently happens when finite difference or finite element methods are being used, the matrix is said to be banded.

If a system of n equations is banded, of bandwidth m, the solution can be found in a time proportional to m^2n rather than n^3, so if $m<<n$ this is a substantial improvement. See [16,50-56] and [7,35] for details.

It should be stressed that if a matrix is not banded, even though it is sparse and regular in some way, it may not be possible to take advantage of its sparseness if we use a direct method. A great deal depends upon precisely how the non-zero elements are distributed.

Sparse and banded systems of equations have been the subject of a large number of papers (for detailed list see [8], as well as books [e.g. 9]). Storage of sparse matrices in a computer is organised in such a way that only the co-ordinates and value of the non-zero coefficients are recorded and special methods have been developed to solve such systems.

We shall say more about this in 4.2. For 'random' sparse matrices the indiscriminate use of Gaussian elimination is not recommended since excessive "fill-in" may occur and the sparseness destroyed. The problem of estimating how fast a "random" sparse matrix will fill-in when Gaussian elimination is carried out is an interesting one. Very roughly the density of non-zeros at first increases fairly slowly as each step of the elimination is carried out until it reaches about 10%, after which it increases very rapidly. For a full analysis and formula see [19]. If the matrix is banded "fill-in" will be moderate; if the matrix is sparse but not banded it may be possible to transform it, by permuting rows and columns, into something like banded form and there are some algorithms which attempt to do this. One technique is to represent the matrix as a directed graph and to reorder the nodes in such a way that groups of adjacent nodes are strongly connected and the

removal of a small number of nodes causes the graph to separate into disjoint subgraphs, the correspondingly reordered matrix will then be in approximately block banded form. For further details and references see [17], 85-146 and p.356. A simple Example illustrating re-ordering is:

Example 5. Limiting fill-in in Gaussian Elimination.

If we use elimination on the matrix

$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad \text{we obtain} \quad \begin{pmatrix} X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & X & X \\ 0 & 0 & 0 & 0 & X \end{pmatrix}$$

whereas if we interchange the first and last rows

$$\begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix} \quad \begin{matrix} \text{and then} \\ \text{eliminate} \\ \text{we get:} \end{matrix} \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ 0 & 0 & 0 & 0 & X \end{pmatrix}$$

If we intend to use an iterative method re-ordering of the rows or columns might affect the convergence. It would be nice if, by suitable reordering of the rows and columns together with appropriate scaling we could guarantee convergencebut I doubt if this is so, certainly I know of no way of doing it.

4.2 Storage of sparse matrices

The general nxr matrix of complex elements requires 2nr words of storage; if the elements are real the storage required is nr words. If the matrix is square and symmetric the storage required can be reduced to $\frac{1}{2}n(n+1)$ words. Whether the matrix is rectangular or square, real or complex if it is sparse very considerable saving in storage is feasible. It hardly needs saying that the fact that a matrix is sparse need not imply that its inverse is sparse so that storage of the inverse of a large sparse matrix may present problems. Two examples of sparse matrices with dense inverses are given in Example 6; the first example is particularly simple, the second is more esoteric but is interesting from several points of view

Example 6

To illustrate that the inverse of a sparse matrix is not necessarily sparse.

(i) The inverse of

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & . & . & . & . \\ -2 & 1 & 0 & 0 & . & . & . & . \\ 0 & -2 & 1 & 0 & . & . & . & . \\ 0 & 0 & -2 & 1 & . & . & . & . \\ . & . & . & . & & & & \\ . & . & . & . & & & & \\ . & . & . & . & & & & \\ . & . & . & . & & & & \end{pmatrix}$$

is easily seen to be

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & . & . & . & . \\ 2 & 1 & 0 & 0 & . & . & . & . \\ 4 & 2 & 1 & 0 & . & . & . & . \\ 8 & 4 & 2 & 1 & . & . & . & . \\ . & . & . & . & & & & \\ . & . & . & . & & & & \\ . & . & . & . & & & & \\ . & . & . & . & & & & \end{pmatrix}$$

(ii) Define A_n as the nxn matrix (a_{ij}) where

$$a_{ij} = (-1)^k \text{ if } i-j = \tfrac{1}{2}k(3k\pm1) \quad (k \geq 0)$$

then A_n is a lower triangular sparse matrix and the non-zero elements are ± 1. The inverse however, $A_n^{-1} = (\alpha_{ij})$ where

$$\begin{aligned} a_{ij} &= p(i-j) \quad \text{for } i \geq j \quad (p(n) \text{ being the partition function}). \\ &= 0 \qquad \text{for } i < j \end{aligned}$$

e.g. for the 7x7 case:

$$A_7 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 & -1 & 1 \end{pmatrix}$$

$$A_7^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 1 & 0 & 0 & 0 \\ 5 & 3 & 2 & 1 & 1 & 0 & 0 \\ 7 & 5 & 3 & 2 & 1 & 1 & 0 \\ 11 & 7 & 5 & 3 & 2 & 1 & 1 \end{pmatrix}$$

In most applications "sparse" matrices contain few non-zero elements in each row, so that, as was said above an nxn matrix would often contain only about kn non-zero terms, where k is a constant. There are however a few applications where the number of non-zero terms in each row increases with n, but slowly; for example the equations for the recurrence formula for computing the values of the partition function illustrated in Example 6 contain approximately $(\frac{8n^{\frac{1}{2}}}{3})$ non-zero terms in the n-th row [13, p.284] - for computational purposes such a matrix should certainly be treated as sparse, particularly if, as in this case, there is a formula which indicates which are the non-zero terms in each row.

When a matrix is sparse the co-ordinates and value of each non-zero element are recorded. Therefore if a real matrix has at most k non-zero elements in each row all its elements can be recorded in 3kn words. It might appear from this that if $k<\frac{1}{3}n$ we will save storage by treating the matrix as sparse; on the other hand however when the elements of a sparse matrix are recorded as triples their manipulation is more time-consuming than when recorded as elements of an array and furthermore if we are performing Gaussian elimination on a sparse matrix we will need access to the elements in particular rows and in particular columns and, to avoid having to search the entire set of triples, it is worth expanding each triple to a quintuple by the addition of two more integer pointers which indicate the addresses of the next non-zero elements in the same row and in the same column. A system such as this has the merit that elements can easily be inserted or deleted as they become non-zero or are eliminated. As a rough guide, therefore, sparse storage techniques should be used only if the $k<\frac{n}{10}$ unless storage is very scarce. A further aspect arises if we are solving the equations on a vector machine since the use of sparse storage techniques implies some level of indirect addressing in an inner loop, and this destroys vectorisation. One solution to this problem is to collect the elements to be processed into a vector, process them using a vectorised loop, and then put them back again. It is here that the Gather and Scatter operations of the Cyber 205 are so valuable; on the CRAY this must be done by software. On a vector processor the advantages of vectorisation are so great that, particularly in the later stages of Gaussian elimination when the sub-matrix is becoming increasingly dense, it

has been found that it is worthwhile switching from sparse matrix to full matrix storage methods - an increase in speed of a factor of 2 having been noted when the density was 20% on a CRAY-1. For further details and discussion see [18].

4.3 Types of Matrices

In most applications two-dimensional matrices are either square (nxn) or rectangular (nxr); the degenerate case when r=1 is equivalent to a vector of n elements; but there are a few esoteric cases where matrices of other 'shapes' occur e.g. L-shaped. Although special storage schemes can be worked out for such cases it is probably best to treat them as rectangular by enclosing them within the smallest rectangular array that will contain them. If however these odd-shaped matrices are also sparse it may be worthwhile to use sparse storage techniques.

A primary objective of many of the computational procedures associated with matrices is to reduce them to some special form which is particularly well suited to other, more powerful, processes. The most important special forms are triangular, tri-diagonal and Hessenberg. "Triangular" in general is appled to nxn matrices (a_{ij}) such that either $a_{ij}=0$ if $i>j$ ("upper-triangular") or $a_{ij}=0$ if $j>i$ ("lower triangular"), "tri-diagonal" implies $a_{ij}=0$ if $|i-j|>1$ and "Hessenberg" matrices have the property that $a_{ij}=0$ if $i>j+1$ ("upper Hessenberg") or $a_{ij}=0$ if $j>i+1$ ("lower Hessenberg").

Transformation of matrices to one or other of these special forms is frequently achieved by means of transformations using orthogonal matrices R_{ij} where R_{ij} is the same as the nxn unit matrix except for four elements viz. $r_{ii}=\cos\theta, r_{jj}=\cos\theta$, $r_{ij}=\sin\theta$, $r_{ji}=-\sin\theta$. It is easily verified that $R_{ij}^T R_{ij}=I$ (i.e. R_{ij} is orthogonal) and that if A is symmetric and θ is suitably chosen the matrix $R_{ij}^T A R_{ij}$ has zero elements in the (i,j) and (j,i) positions.

4.4 Operations on matrices

Perhaps the most fundamental operation of all in any set of matrix and vector manipulation routines is that of forming the inner product of two vectors. If this operation can be carried out fast and accurately in a subroutine the benefits will be noticeable everywhere. For accuracy it is usual to carry out the accumulation of inner products in double-length precision whilst for speed a technique such as that of Winograd, referred to in Dr. Zacharov's lectures, can be used if the formation of inner products is part of a larger problem, such as matrix multiplication.

In general any package of programs for solving problems involving systems of linear equations and matrices will benefit from having thoroughly tested subroutines to perform the following tasks: interchange two rows (or columns) of A, find the largest element in a given row (column) of A, find

the largest element in a given row (column) of A, find the largest element in A, find the largest off-diagonal element of A, scale the elements of a row (column) of A so that the largest element is unity, multiply a matrix by a vector. In all cases the subroutine should return parameters indicating whether exceptional cases have been encountered e.g. that there are two or more equal "largest" elements, or that the matrices to be multiplied have incompatible dimensions, etc. Furthermore, a set of well-chosen test matrices, such as those given in [14], should be used as test data before matrix packages are released to the users at large.

The major operations on matrices are (1) inversion, (2) finding one or more eigenvalues. Techniques for achieving these aims are well described in a vast literature of papers and books; it would take far too long to describe them in detail and it would also be pointless with so many good accounts readily available. What I shall attempt to do therefore in the next two sections is to summarise the main methods for matrix inversion and finding eigenvalues, give references to detailed accounts of these methods and indicate which methods are best suited to a variety of situations.

5. MATRIX INVERSION

The problem of the inversion of an nxn matrix is essentially the same as the problem of solving a system of n linear equations with n right-hand sides for if A is the matrix to be inverted then we are seeking a matrix X such that

$$AX = I_n \tag{5.1}$$

(I_n being the nxn unit matrix) and if the elements of X are (x_{ij}) then the k-th column of X, $\underline{x}_k$ say, can be found by solving the system of n linear equations

$$A\underline{x}_k = \underline{b}_k \tag{5.2}$$

where $\underline{b}_k$ is a column vector having all its elements equal to zero with the exception of the k-th which is equal to 1.

We see therefore that we can find the inverse of a matrix by using any appropriate methods for solving systems of linear equations such as those described in Section 3. There is not much to choose in efficiency as measured by the number of operations required, between any of the direct methods unless the matrix is symmetric, when a method which exploits the symmetry, such as Choleski factorisation, is to be preferred. The Doolittle method is slightly superior to either Gaussian elimination or Gauss-Jordan in that it requires (n^3-n) multiplications compared to (n^3-1) required by the other two methods. It is a curious fact that although the Gauss-Jordan is 50% less efficient than the method of Gaussian elimination for a single equation the two methods require exactly the same number of multiplications, divisions, and additions/subtractions to compute the inverse of a matrix.

Indirect methods would not normally be used for finding the inverse of a matrix or, indeed for solving a system with many right-hand sides, since although the convergence criteria are independent of the r.h.s. of the equations a separate computation would still be necessary for each r.h.s. Indirect methods for solving equations are best when the equations are diagonally dominant and sparse. Note, incidentally, that although the inverse of a lower (upper) triangular matrix is also a lower (upper) triangular matrix sparseness as we have noted already is not a property which is necessarily preserved under inversion; the matrix associated with the formula for the partition function mentioned in Section 4.2 is lower triangular and sparse with coefficients 0,+1 or -1 but its inverse has the positive integer p(i-j) as its (i,j) element (i>j) where p(m) denotes the number of partitions of m as the sum of positive integers and

$$p(m) \sim \frac{1}{4m\sqrt{3}} \exp \pi\sqrt{\frac{2m}{3}}$$

as $m \to \infty$. Thus the inverse of this particular sparse matrix is dense and has elements which grow very rapidly. On the other hand a matrix which has exactly one non-zero element in each row and in each column has an inverse which is of the same type (in fact if $a_{ij}=\alpha(\neq 0)$ in A then A^{-1} has α^{-1} as its (j,i)-th element).

A brief mention should be made of a class of problems where we have a very large sparse matrix and require to find only a specific subset of the elements of its inverse. It was in connection with a problem of this type that von Neumann and Ulam invented the Monte Carlo method which has the merit that it enables us to calculate approximate values of any particular elements in the inverse fairly easily but has the disadvantage that the computer time required for even a moderate increase in accuracy may be enormous. For an early description of the method see [20]; for further details and references see [7, pages 70-75]. A different approach, based upon a suggestion of Takahashi, has been proposed by Erisman and Tinney and offers significant advantages if relatively few elements of the inverse are required; for details see [21].

In the special case where all the desired elements lie in one, or a few, columns they can, of course, be obtained by simply solving $A\underline{x} = \underline{b}$ and setting $\underline{b}$ equal to the appropriate column(s) of the unit matrix.

6. EIGENVALUES

The need to compute eigenvalues arises in a surprisingly wide range of problems. In these lectures we shall only be concerned with the algebraic eigenvalue problem viz. given an nxn matrix A to find one or more of the numbers λ and vectors $\underline{x}(\neq\underline{0})$ such that

$$A\underline{x} = \lambda\underline{x} \tag{6.1}$$

An eigenvalue problem also occurs in differential equations e.g. the differential equation

$$p(x)y'' + q(x)y' = \lambda g(x)y \tag{6.2}$$

given $y(0) = 0$, $y'(0) = 0$ has the trivial solution $y(x)\equiv 0$ but it may also have non-trivial solutions for certain values of λ.

The values of λ in both (6.1) and (6.2) are called eigenvalues; the vectors $\underline{x}$ in (6.1) are called eigenvectors and the solutions $y(x)$ in (6.2) are called eigenfunctions. When (6.2) is solved numerically it is usually converted to an algebraic form in which case methods for the solution of (6.1) are relevant. The literature associated with (6.1) is vast, as is indicated by the most important book on the subject [6], by Wilkinson, which though published in 1965 contains over 600 pages. As for the range of applications of eigenvalues a glimpse is provided by Gourlay and Watson [15] who illustrate with examples from mechanics, information science, economics and physics.

The algebraic eigenvalue problem for an nxn matrix A is the evaluation of those values of λ for which the system of n linear equations

$$A\underline{x} = \lambda\underline{x} \tag{6.3}$$

has a solution $\underline{x} = (x_1, x_2 .., x_n)$ with at least one element x_i (say) $\neq 0$. For (6.3) to have a non-trivial solution x we must have

$$\det(A-\lambda I) = 0 \tag{6.4}$$

and since (6.4) is equivalent to finding the n roots (not necessarily all distinct) of the characteristic polynomial we already have a theoretical method for solving the problem. However, as in the case of finding the determinant of a matrix, this is not a method to be recommended unless n is very small ($n\leqslant 3$?).

6.1. Numerical determination of eigenvalues

An nxn matrix has n eigenvalues, some of which may be equal, some may be complex; let them be $\lambda_1, \lambda_2 \ldots\lambda_n$, numbered so that $|\lambda_1|\geq|\lambda_2|\geq\ldots\geq|\lambda_n|$; λ_1 is then called the dominant eigenvalue. A variety of methods have been developed for finding eigenvalues; some methods apply to general matrices, others only to symmetric matrices, others to tri-diagonal or Hessenberg matrices; some methods find only the dominant eigenvalue, others find all n eigenvalues. It follows that the would-be user must begin by answering certain questions: (i) is the matrix A known to have any special properties

such as symmetry, being banded, or positive-definite etc, (ii) do I want only the dominant eigenvalue, or the eigenvalue closest to a particular value, or some subset of the eigenvalues, or all of them?, (iii) are the eigenvectors also required?

6.2. Similarity transformations

Some of the most important methods for finding eigenvalues reply upon the fact that if R is a non-singular nxn matrix then the matrix $R^{-1}AR$ has the same eigenvalues of A; for if $A\underline{x} = \lambda\underline{x}$ then $(R^{-1}AR)(R^{-1}\underline{x}) = \lambda(R^{-1}\underline{x})$ - note however that the eigenvector associated with λ is no longer $\underline{x}$, but $R^{-1}\underline{x}$. The matrices A and $R^{-1}AR$ are said to be similar, and the transformation $A \to R^{-1}AR$ is called a similarity transformation.

6.3. Approximate location of eigenvalues

It is often helpful if we can locate the eigenvalues approximately before applying more refined procedures. The most useful general theorem, which applies to any nxn matrix is:

Gerschgorin's Theorem Let A be an nxn matrix and let C_i (i=1,2,...,n) be the discs with centres a_{ii} and radii

$$R_i = \sum_{\substack{k=1 \\ k \neq i}}^{n} |a_{ik}|.$$

Let D denote the union of the discs C_i. Then all the eigenvalues of A lie in D.

The theorem also applies to A^T, which has the same eigenvalues of A and if A is asymmetric this provides further information about the location of the eigenvalues. For a proof of the theorem see [, p22].

A refinement of the theorem may provide some more information. Let $T_1, T_2, \ldots, T_m (m \leqslant n)$ be the non-overlapping Gerschgorin domains defined by the Gerschgorin discs $C_1, C_2, \ldots, C_n$. Then the number of eigenvalues inside T_i is the same as the number of overlapping discs which make up T_i. (See [3], p103.)

Example 7. The use of Gerschgorin's Theorem

We apply the (refined version of) the theorem to the matrix

$$A = \begin{pmatrix} 2.00 & 0.90 & 0.10 \\ 1.05 & 4.50 & 0.25 \\ 0.00 & 0.20 & 6.20 \end{pmatrix}$$

The discs are: $C_1 : |x-2.00| \leqslant 1.00$

$C_2 : |x-4.50| \leqslant 1.30$

$C_3 : |x-6.20| \leqslant 0.20$

These discs do not overlap and so there are 3 separated domains T_1, T_2, T_3 and each contains one eigenvalue. The eigenvalues must therefore all be real, since if there were a complex conjugate pair they would lie in the same disc (which is symmetric about the x-axis). Hence we can say $6.00 < \lambda_1 < 6.40$, $3.20 < \lambda_2, < 5.80$, $1.00 < \lambda_3 < 3.00$. If we apply the theorem to A^T we find that we can improve the second inequality to $3.40 < \lambda_2 < 5.60$. In fact $\lambda_1 \doteq 6.23$, $\lambda_2 \doteq 4.805$, $\lambda_3 \doteq 1.665$.

In the case when A is a tri-diagonal matrix the Sturm sequence technique can be used to locate eigenvalues as accurately as we wish; this technique is covered in Section 6.6.

6.4 Methods for finding the dominant eigenvalue

In many applications only the eigenvalue of largest modulus ("the dominant eigenvalue") is required in which case the Power Method, which is easy to program, is perhaps the most useful general method.

6.4.1 The Power Method Let A be an nxn real matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$ where $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. The Power Method is used to find λ_1 and its corresponding eigenvector $\underline{x}$. We begin by choosing an arbitrary starting approximation to $\underline{x}$, $\underline{x}_1$ ($\neq 0$) say; it is quite common to take $\underline{x}_1$ as (1,1,1..,1) unless we have some reason for choosing otherwise. We now proceed iteratively as follows:

$$\text{Let } \underline{y}_k = A\underline{x}_k \qquad k = 1, 2, .. \tag{6.5}$$

and

$$\underline{x}_{k+1} = \frac{\underline{y}_k}{||\underline{y}_k||_\infty} \tag{6.6}$$

In (6.6) division by $||\underline{y}_k||_\infty$ ensures that the largest element in $\underline{x}_{k+1}$ is unity, and so prevents the elements becoming large. The iteration continues until $||y_k||_\infty$ and $\underline{x}_k$ stabilise, they are then taken as λ_1 and $\underline{x}$.

The process converges with a speed proportional to $|\lambda_2/\lambda_1|^n$ - if this ratio is small convergence is fast; if it is close to 1 convergence is slow, if it is equal to 1 we must modify the method (see [6], Chapter 9).

If the iterations are converging very slowly we can use Aitken's method to accelerate the convergence (see Section 3.4.1).

Example 8. Find the dominant eigenvalue of

$$\begin{pmatrix} 2.00 & 0.90 & 0.10 \\ 1.05 & 4.50 & 0.25 \\ 0.00 & 0.20 & 6.20 \end{pmatrix}$$

by the Power Method and use Aitken's method to accelerate convergence.

Solution	k	x_k^T		$\|\|Y_k\|\|_\infty$
	0	1	1	1
	1	0.469	0.906	6.4
	2	0.290	0.755	6.381
	3	0.214	0.622	6.351
	4	0.172	0.518	6.324
	5	0.144	0.438	6.304

Convergence is clearly very slow, after 10 iterations we get:

10	0.083	0.249	1	6.254

If we use the 3rd, 4th and 5th iterations Aitken's extrapolation yields the revised estimate $\lambda \doteq 6.246$, which is better than the value obtained at the 10th iteration.

In this case $\lambda_2/\lambda_1 \doteq 0.77$, which accounts for the slow convergence; shifting the origin by 3 cuts the iterations required by about 40%.

It may also be possible to accelerate the convergence of the Power Method by "shifting the origin". Suppose that the eigenvalues of A are all real and are arranged in order $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ and $\lambda_1 > |\lambda_n|$; let c be a real constant then the matrix A-cI has eigenvalues $\lambda_1 - c$, $\lambda_2 - c$, .., $\lambda_n - c$. The speed of convergence of the Power Method applied to A-cI is linear with a factor

$$\text{Max} \left(\frac{|\lambda_2 - c|}{|\lambda_1 - c|} , \frac{|\lambda_n - c|}{|\lambda_1 - c|} \right) \tag{6.7}$$

and it is easily seen that the optimal choice for c is

$$c = \frac{1}{2} (\lambda_2 + \lambda_n) \tag{6.8}$$

when the two ratios in (6.7) are equal. In the absence of any information about λ_2 and λ_n a suitable value for c can only be determined experimentally.

If A is symmetric the Rayleigh-Ritz Quotient can be used to give a more rapidly converging sequence of approximations to λ_1 since in the notation of (6.5) and using inner products

$$\frac{(y_k, x_k)}{(x_k, x_k)} \rightarrow \lambda_1 \tag{6.9}$$

where $(y_k, x_k) = \sum_{i=1}^{n} y_{i,k} \, x_{i,k}$ (6.10)

and this process converges linearly but with a ratio $(\lambda_2/\lambda_1)^2$.

6.4.2 Finding the subdominant eigenvalues If we wish to find one or more, but not all, of the subdominant eigenvalues there are two techniques available based upon the Power Method. The first technique is known as "Deflation"; having found the dominant eigenvalue, λ_1, and corresponding eigenvector $\underline{x}_1$, of the nxn matrix A we transform A_1 into another matrix A having the same eigenvalues but having the form

$$A_1 = \begin{pmatrix} \lambda_1 & c_1 \\ 0 & B_1 \end{pmatrix} \tag{6.11}$$

where B_1 is an (n-1) x (n-1) matrix, and c_1 is an (n-1) element row vector. Since the eigenvalues of B_1 are $\lambda_2,..,\lambda_n$ we can now use the Power Method to find λ_2, and so on. The details, including those relating to recovery of the eigenvectors of A, can be found in [15, pp. 47-50].

The second method enables us to compute the eigenvalue closest to any given value, m, say. The matrix

$$A-mI \tag{6.12}$$

has eigenvalues λ_1-m, λ_2-m, .., λ_n-m and so the absolutely smallest eigenvalue of A-mI is that eigenvalue of A closest to m, λ_j, say. It follows that the largest eigenvalue of the inverse

$$(A-mI)^{-1} \tag{6.13}$$

is λ_j^{-1}; and so if we apply the Power Method to (6.13) we will obtain λ_j^{-1}, and hence λ_j. For a full discussion of this method see [15, 56-62].

6.5 Finding all eigenvalues

6.5.1 Jacobi's method This method can be applied to symmetric matrices; the basis of the method is the construction of a series of orthogonal matrices R_{ij} which are used to transform A through a sequence A_1, A_2,... defined by

$$A_{s+1} = R_{ij} A_s R_{ij}^T \tag{6.14}$$

so that each A_s has the same eigenvalues as A and yet the sum of the squares of the off-diagonal elements of A_{s+1} are less than those of A_s; ultimately A_s tends to a diagonal matrix and its diagonal elements are the eigenvalues of A. Construction of the matrix R_{ij} at each iteration is quite easy; at the s-th stage suppose that $a_{ij}^{(s)}$ is the largest off-diagonal element in the upper triangle of A_s. Let θ be that angle in $<-\pi/4, \pi/4>$ such that

$$\text{Tan } 2\theta = \frac{2a_{cj}^{(s)}}{(a_{ii}^{(a)} - a_{jj}^{(s)})} \tag{6.15}$$

Take R_{ij} as a matrix U which is the same as the unit matrix except that

$u_{ii} = \cos\theta$, $u_{jj} = \cos\theta$, $u_{ij} = \sin\theta$, $u_{ji} = -\sin\theta$ where θ is defined by (6.15). Then apply (6.14) to the calculation of A_{s+1}; the (i.j)-th and (j,i)-th elements of A_{s+1} will now be zero and furthermore the sum of the squares of the off-diagonal elements of A_{s+1} will be $2(a_{ij})^2$ less than those of A_s and the sum of the squares of the diagonal elements will be $2(a_{ij})^2$ greater than those of A_s.

For further details see [15, Chapter 7].

Note, incidentally, that if A is a symmetric matrix then

$$\sum_{i=1}^{n} \lambda_i^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2 \qquad (6.16)$$

- this is deducible from the remarks just made about Jacobi's method but is easy to prove directly since the eigenvalues of A^2 are $\lambda_1^2, \lambda_2^2, .., \lambda_n^2$ and the trace of A^2 is the expression on the r.h.s. of (6.16). This identity enables us to put lower and upper bounds on λ_1^2.

Example 9 Find the eigenvalues of

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 4 & 2 & 5 \\ 1 & 5 & 3 \end{pmatrix}$$

by Jacobi's method.

Solution (We can get bounds on λ_1, for $\lambda_1^2 + \lambda_2^2 + \lambda_3^2 = 98$ hence $\lambda_1^2 \leqslant 98$, and $3\lambda_1^2 \geqslant 98$ so $5.7 < |\lambda_1| < 9.9$). In the Jacobi method (i) the largest off-diagonal element is $a_{23} = 5$, so we choose $\text{Tan } 2\theta = \frac{2\times5}{2-3} = -10$, which gives $\theta = -42^{\circ}9'$ and the rotation matrix is

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0.741 & -0.671 \\ 0 & 0.671 & 0.741 \end{pmatrix}$$

leading to

$$A_1 = \begin{pmatrix} 1 & 2.293 & 3.425 \\ 2.293 & 12.522 & -0.003 \\ 3.425 & -0.003 & 7.519 \end{pmatrix}$$

- note the effect of rounding errors (working to 3 d.p.). After 3 more transformations of this type we get

$$A_4 = \begin{pmatrix} 0.850 & -0.028 & 0.478 \\ -0.028 & -3.883 & 0.000 \\ 0.478 & 0.000 & 9.009 \end{pmatrix}$$

The diagonal elements are now close to λ_1, λ_2, λ_3 which (as given by the

roots of the original characteristic equation $\lambda^3-6\lambda^2-31\lambda+29 = 0$) are 9.07, -3.89 and 0.82 to 2 d.p.

Two further points about the Jacobi method should be noted. The first is a bad point: an element which has been reduced to zero by one transformation may become non-zero as the result of a subsequent transformation. In consequence the convergence may be slow and, in any case, we cannot predict the number of transformations required to complete the reduction to diagonal form. The second is a good point: if we want the eigenvectors as well as the eigenvalues the Jacobi method yields them automatically for at the end of the Jacobi process we have reduced A to a purely diagonal matrix D as the result of a series of transformations of the type (6.14), thus we have effectively found a matrix R such that

$$RAR^T = D \tag{6.17}$$

and so

$$AR^T = R^TD \tag{6.18}$$

Since $R^{-1} = R^T$ and it follows that the columns of R^T are the eigenvectors of A. So, if we require the eigenvectors we should compute R^T as each transformation matrix R_{ij} is obtained.

6.5.2 Reduction to tri-diagonal form

If we do not wish to find all the eigenvectors of a matrix the Jacobi method loses some of its attraction and this combined with a possible slow convergence for large matrices, led to the development of methods which reduced the eigenvalue problem to one which could be more simply tackled in a finite number of steps. The two principal methods, due to Givens (1954) and Householder (1958) both reduce A to a tri-diagonal matrix if A is symmetric and to a Hessenberg matrix if A is not symmetric. In the symmetric case the Givens method uses rotation matrices to annihilate all the elements except those on the main diagonal and its immediate neighbours two at a time; the Householder method uses elementary Hermitian matrices to annihilate all the elements of a column (and corresponding row) in a single transformation. The Householder method is significantly more efficient in general. For details of both processes see [15, 71-78]. It has been pointed out, however, that if A is sparse these transformations may result in a lot of "fill-in", leading to a substantial amount of additional work. For discussion of the problem of finding eigenvalues of sparse matrices see [17, 130-140].

6.6 Finding the eigenvalues of a tri-diagonal matrix

When the matrix has been reduced to a tri-diagonal form our choice of method for finding the eigenvalues depends upon whether we want a few of the eigenvalues or all of them. In the former case the Sturm sequence property can be used, with the method of bisection, to obtain specific eigenvalues to any precision. The Sturm sequence for the tri-diagonal matrix, T, of order n shown below

$$T = \begin{pmatrix} a_1 & b_1 & 0 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & 0 & \cdots & \\ 0 & b_2 & a_3 & b_3 & \cdots & \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & 0 \\ \cdot & \cdot & & & & b_{n-1} \\ 0 & 0 & \cdot & & b_{n-1} & a_n \end{pmatrix} \tag{6.19}$$

is the sequence of (n+1) polynomials $p_0(\lambda)$, $p_1(\lambda)$, ... $p_n(\lambda)$ defined by

$$p_0(\lambda) = 1, \; p_1(\lambda) = a_1 - \lambda$$

and, for $r \geqslant 2$: $p_r(\lambda) = (a_r - \lambda)p_{r-1}(\lambda) - b_{r-1}^2 p_{r-2}(\lambda)$ (6.20)

It is easy to see that $p_r(\lambda)$ is the determinant of the r-rowed principal minor of $(T-\lambda I)$ and, in particular

$$p_n(\lambda) = \det(T-\lambda I) \tag{6.21}$$

so that the zeros of $p_n(\lambda)$ are the eigenvalues of T. The important property of these polynomials is summarised by:

Theorem Let $s(\lambda)$ denote the number of agreements in sign between successive members of the sequence $p_0(\lambda)$, $p_1(\lambda)$, ..., $p_n(\lambda)$ where if $p_r(\lambda)$ is zero its sign is taken as opposite to that of $p_{r-1}(\lambda)$; then $s(\lambda)$ is equal to the number of eigenvalues of λ that are strictly greater than λ.

This theorem can be used to locate the eigenvalues approximately and when an interval I_r is known to contain λ_r the process of bisection can be applied to determine λ_r as accurately as required. The bisection process is slow, $3\frac{1}{3}$ iterations being required on average to improve precision by one decimal place, but converence is certain.

Example 10 By applying the Sturm sequence to the matrix

$$\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

with λ=0,1,2 and 3 obtain bounds on the 4 eigenvalues.

Solution $p_o(\lambda) = 1,\ p_1(\lambda) = (2-\lambda)$

$$p_2(\lambda) = (2-\lambda)p_1(\lambda) - p_o(\lambda)$$

$$p_3(\lambda) = (2-\lambda)p_2(\lambda) - p_1(\lambda)$$

$$p_4(\lambda) = (2-\lambda)p_3(\lambda) - p_2(\lambda)$$

and so (i) at $\lambda = 0$ the values are 1,2,3,4,5 giving signs +++++
hence s(0) = 4

(ii) at $\lambda = 1$ the values are 1,1,0,-1,-1 which counts as ++---
hence s(1) = 3

(iii) at $\lambda = 2$ the values are 1,0,-1,0,1 which counts as +--++
hence s(2) = 2

(iv) at $\lambda = 3$ the values are 1,-1,0,1,-1 which counts as +-++-
hence s(3) = 1

It follows from these results that there is one eigenvalue in each of the intervals (0,1), (1,2) (2,3) and (3, ∞). In fact the eigenvalues are

0.382, 1.382, 2.618 and 3.618 to 3 d.p.

If all the eigenvalues of the tri-diagonal system are required it is better to use a method such as the QR method, introduced by Francis in 1961. For a full description and analysis of this method see [15, Chapter 8]. The idea underlying the method, due to Rutishauser (1958) is that A can be factorised

$$A = QR \tag{6.22}$$

where R is upper triangular and Q is orthogonal. If we write $A_1 = A$ and define a sequence of matrices A_m, Q_m, R_m by

$$A_m = Q_m R_m,\ A_{m+1} = R_m Q_m,\ m=1,2... \tag{6.23}$$

then the sequence A_m will converge either to a triangular matrix with the eigenvalues of A on its diagonal or to a near-triangular matrix from which the eigenvalues may be easily calculated. Convergence may be slow so a technique called shifting is usually employed to accelerate it.

The QR method can also be applied to Hessenberg matrices; for a detailed account, including a program (in Algol) see [16, 359-371].

If the eigenvectors are required the most useful method is that of inverse iteration for a detailed description and program for which see [16, 418-439].

REFERENCES

[1] Churchhouse, R.F. "Computer Arithmetic and the Failure of the Associative Law" I.M.A. Bulletin, 16 (1980), 210-214.

[2] Erskine, G.A. "Computer arithmetic," Proc. 1976 CERN School of Computing, 191-200.

[3] Churchhouse, R.F. "Survey of methods of numerical approximation" Proc. 1980 CERN School of Computing, 78-105.

[4] Ralston, A. "A first course in numerical analysis," McGraw Hill, 1965.

[5] Wilkinson, J.H. "Error Analysis of Direct Methods of Matrix Inversion," J.ACM 8, 281-330 (1961).

[6] Wilkinson, J.H. "The algebraic eigenvalue problem," Clarendon Press, Oxford, 1965.

[7] Westlake, Joan. "A handbook of numerical matrix inversion and solution of linear equations," John Wiley and Sons, 1968.

[8] Duff, I.S. "A Survey of sparse matrix research," Report HL 76/485 (1976), A.E.R.E., Harwell, U.K.

[9] Reid, J. K. (Ed). "Large sparse sets of linear equations," Academic Press, 1971.

[10] Young, D.M. "Iterative solution of large linear systems," Academic Press, 1971.

[11] Isaacson, E. and Keller, H. "Analysis of Numerical Methods," Wiley, 1976.

[12] Churchhouse R.F. (Ed.) "Handbook of Applicable Mathematics," Vol.III (Numerical Methods), Wiley, 1981.

[13] Hardy, G.H. and Wright, E.M. "Introduction to the theory of numbers," Oxford University Press, 4th Edition (1975).

[14] Gregory, R.T., Karney, D.L. "A collection of matrices for testing computational algorithms." Wiley - Interscience (1969).

[15] Gourlay, A.R. and Watson, G.A. "Computational methods for matrix eigenproblems," Wiley (1975).

[16] Wilkinson, J.H. and Reinsch, C. ed. "Handbook for Automatic Computation" Vol. LL, Linear Algebra; Springer-Verlag, New York, 1971.

[17] Jacobs, D. (ed.) "The State of the Art in Numerical Analysis" Academic Press, 1977.

[18] Duff, I.S. "The solution of sparse linear equations on the Cray-1", Report HL 82/1370 (1982) A.E.R.E., Harwell, U.K.

[19] Duff, I.S. "On the number of nonzeros added when Gaussian elimination is performed on sparse random matrices", Math.Comp. 28 (1974), 219-230.

[20] Forsythe, G.E.,Leibler, R.A. "Matrix Inversion by a Monte Carlo Method", M.T.A.C., 5 (1953), 127-129 and 6 (1954), 55.

[21] Erisman A.M. and Tinney, W.F. "On Computing certain Elements of the Inverse of a Sparse Matrix", C.A.C.M. 18 (1975), 177-179.