MATRIX COMPUTATION

E. Edberg

Lectures given in the
Academic Training Programme of CERN 1975-1976

GENEVA
1976
Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l'autorisation écrite du Directeur général du CERN, titulaire du droit d'auteur. Dans les cas appropriés, et s'il s'agit d'utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés. Cependant, le CERN se réserve le droit de s'opposer à toute revendication qu'un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrit dans le présent document.

© Copyright CERN, Genève, 1976

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN. However, permission will be freely granted for appropriate non-commercial use. If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.

CERN - Service d'Information scientifique - RD/195 - 2000 - décembre 1976
ABSTRACT

These lectures present appropriate numerical methods for the solution of systems of linear equations, linear least-squares problems, and matrix eigenvalue problems. Particular emphasis is given to the way in which the special properties of the problem and the choice of method may affect the size of the errors. The chapter on linear systems deals mainly with elimination methods. For the linear least-squares problem, the transformation of matrices to upper triangular form by Householder transformations is discussed, and the singular value decomposition and the Moore-Penrose generalized inverse are introduced. The QR method and inverse iteration are presented for the eigenvalue problem $Ax = \lambda x$. The treatment is kept at a level suitable for non-specialists.
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER I: SOLUTION OF SYSTEMS OF LINEAR EQUATIONS</td>
<td>1</td>
</tr>
<tr>
<td>1. NOTATIONS</td>
<td>1</td>
</tr>
<tr>
<td>2. SOME MATHEMATICAL BACKGROUND</td>
<td>1</td>
</tr>
<tr>
<td>3. GAUSSIAN ELIMINATION</td>
<td>2</td>
</tr>
<tr>
<td>4. PIVOTING</td>
<td>7</td>
</tr>
<tr>
<td>4.1 Formalism of partial pivoting</td>
<td>7</td>
</tr>
<tr>
<td>5. SCALING OF THE COEFFICIENT MATRIX</td>
<td>8</td>
</tr>
<tr>
<td>6. THE CROUT AND DOOLITTLE METHODS</td>
<td>9</td>
</tr>
<tr>
<td>7. CHOLESKY DECOMPOSITION FOR REAL SYMMETRIC, STRICTLY POSITIVE MATRICES</td>
<td>10</td>
</tr>
<tr>
<td>8. ERROR ANALYSIS</td>
<td>12</td>
</tr>
<tr>
<td>8.1 Backwards error analysis for numerical methods</td>
<td>13</td>
</tr>
<tr>
<td>8.2 Some error bounds</td>
<td>13</td>
</tr>
<tr>
<td>9. ILL- AND WELL-CONDITIONED PROBLEMS; CONDITION NUMBER FOR INVERSION</td>
<td>14</td>
</tr>
<tr>
<td>10. PRACTICAL WAYS OF DETECTING ILL-CONDITIONING</td>
<td>15</td>
</tr>
<tr>
<td>10.1 Test on pivots</td>
<td>15</td>
</tr>
<tr>
<td>10.2 Iterative improvement</td>
<td>17</td>
</tr>
<tr>
<td>11. ITERATIVE METHODS</td>
<td>18</td>
</tr>
<tr>
<td>CHAPTER II: LINEAR LEAST SQUARES PROBLEMS</td>
<td>20</td>
</tr>
<tr>
<td>1. HOUSEHOLDER TRANSFORMATIONS</td>
<td>20</td>
</tr>
<tr>
<td>2. THE LINEAR LEAST SQUARES PROBLEM</td>
<td>22</td>
</tr>
<tr>
<td>3. THE QU-DECOMPOSITION</td>
<td>23</td>
</tr>
<tr>
<td>4. SOLVING THE LINEAR LEAST SQUARES PROBLEM BY QU-FACTORIZATION</td>
<td>25</td>
</tr>
<tr>
<td>5. OBTAINING A PRESCRIBED ACCURACY</td>
<td>26</td>
</tr>
<tr>
<td>6. THE CONDITION OF THE LEAST SQUARES PROBLEM</td>
<td>27</td>
</tr>
<tr>
<td>7. OTHER METHODS</td>
<td>28</td>
</tr>
<tr>
<td>7.1 Gram-Schmidt orthogonalization</td>
<td>28</td>
</tr>
<tr>
<td>7.2 The singular value decomposition and the Moore-Penrose generalized inverse</td>
<td>28</td>
</tr>
</tbody>
</table>
CHAPTER III: THE EIGENVALUE PROBLEM $Ax = \lambda x$

1. SOME MATHEMATICAL BACKGROUND
   30

2. THE CONDITIONING OF THE EIGENVALUE PROBLEM
   32

3. THE QR METHOD
   35
   3.1 Similarity transformations; Hessenberg matrices
   35
   3.2 Brief description of the QR algorithm
   38

4. COMPUTATION OF EIGENVECTORS
   40
   4.1 The power method
   40
   4.2 Inverse iteration for eigenvectors (Wielandt iteration)
   41

5. THE COMPUTATION OF THE SINGULAR VALUE DECOMPOSITION
   41

REFERENCES
   43

SUBJECT INDEX
   45
CHAPTER I

SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

In this chapter we will present some useful algorithms for the solution of systems of linear equations. We will discuss the numerical properties of the methods, such as propagation of rounding errors and the number of arithmetic operations required. The distinction between a "bad" method and a "bad" problem will be pointed out, and some ways of detecting ill-conditioning will be considered.

1. NOTATIONS

Consider a system of \( n \) linear equations in \( n \) unknowns \( x_1, x_2, \ldots, x_n \):

\[
\begin{align*}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{align*}
\]

In matrix notation, this can be written as

\[
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
&\vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{bmatrix}
=
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{bmatrix}
\]

or shorter,

\[
Ax = b,
\]

where \( A \) is the \( n \times n \) coefficient matrix, \( x \) is the unknown vector with \( n \) components, and \( b \) is the right-hand-side vector with \( n \) components. This notation will be used throughout this chapter. In general, matrices will be denoted by capitals and vectors by lower case letters.

2. SOME MATHEMATICAL BACKGROUND

A unique solution \( x \) to \( Ax = b \) exists if and only if \( A \) is non-singular; that is, when the determinant of \( A \), \( \det(A) \), is non-zero.
If \( \det(A) = 0 \), either no solution exists or there are infinitely many solutions. Since computers do not perform exact arithmetic, a small determinant will cause numerical problems on a computer.

If \( A \) is non-singular, the inverse \( A^{-1} \) exists and the unique solution of the system can formally be written as \( x = A^{-1} b \).

**IMPORTANT NOTE:** If we only want to find the solution of a linear system of equations, the inverse \( A^{-1} \) is *not* explicitly needed. If we compute \( A^{-1} \) and then perform the matrix-vector multiplication \( A^{-1} x \), we do about three times as much work as is actually needed.

3. **GAUSSIAN ELIMINATION**

Gaussian elimination is nothing but the elimination method that most people would use if they had to solve a system of linear equations by hand, and which has been taught in schools long before computers came into use. With computers came the possibility of solving larger and larger systems, and this made it necessary to study the influence of accumulated round-off errors, and to modify this method (and others) in order to improve the numerical behaviour. Also, the method had to be formalized so that it could be coded in a programming language.

Let us first look at a very simple example of Gaussian elimination and see how it is written in matrix notation:

**Example 1**

\[
\begin{align*}
10 \ x_1 + 5 \ x_2 &= 15 \\
2 \ x_1 + 3 \ x_2 &= 5
\end{align*}
\]

The first equation (row) is kept unchanged and \( (2/10) \times \) row 1 is subtracted from row 2.

The resulting system is

\[
\begin{align*}
10 \ x_1 + 5 \ x_2 &= 15 \\
2 \ x_2 &= 2
\end{align*}
\]

The second equation gives \( x_2 \) and \( x_1 \) can then be computed from the first equation. This is called **back-substitution**.

What we did was to modify the coefficient matrix as well as the right-hand-side vector. In matrix notation this can be written as

\[
\begin{pmatrix}
1 & 0 \\
2 & 1
\end{pmatrix}
\begin{pmatrix}
10 & 5 & 15 \\
2 & 3 & 5
\end{pmatrix}
= \begin{pmatrix}
10 & 5 & 15 \\
0 & 2 & 2
\end{pmatrix}
\]

* * *
For a general \( n \times n \) system, Gaussian elimination can be formalized as follows:

**Step 1**

Form the product

\[
\begin{bmatrix}
1 & 0 & 0 & \cdots & 0 \\
-\frac{a_{21}}{a_{11}} & 1 & 0 & \cdots & 0 \\
-\frac{a_{31}}{a_{11}} & 0 & 1 \\
\vdots & \vdots & \ddots \\
-\frac{a_{n1}}{a_{11}} & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\
a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\
a_{31} & \cdots & \cdots & \cdots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn} & b_n
\end{bmatrix} = M_1(A|b).
\]

The premultiplication of \( M_1 \) onto \((A|b)\) has the following effect:

- the first row of \((A, b)\) is left unaltered,
- \(\frac{a_{21}}{a_{11}} \times \text{row 1} \) is subtracted from row 2,
- \(\frac{a_{31}}{a_{11}} \times \text{row 1} \) is subtracted from row 3,
- \( \vdots \)
- \(\frac{a_{n1}}{a_{11}} \times \text{row 1} \) is subtracted from row \(n\).

(Note that a premultiplication onto a matrix can always be regarded as replacing the rows in the matrix by a linear combination of the old rows.)

The new matrix \(((A|b)^{(2)})\) = \(M_1(A|b)\) has the form

\[
\begin{bmatrix}
a_{11} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots & a_{1n}^{(2)} & b_1^{(2)} \\
0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\
0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} & b_3^{(2)} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)}
\end{bmatrix}
\]

**Step 2**

We want to keep the first and the second row and produce zeros in positions \((3,2), (4,2), \ldots, (n,2)\). This corresponds to the elimination of \(x_2\) from equations 3, 4, \ldots, \(n\). It can be achieved with the following transformation:
\[
\begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 0 & & \\
0 & -\frac{a_{22}^{(2)}}{a_{22}^{(2)}} & 1 & & \\
0 & -\frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 0 & 1 & \\
\vdots & \vdots & \vdots & \ddots & \\
0 & -\frac{a_{n2}^{(2)}}{a_{22}^{(2)}} & 0 & \ldots & 1
\end{bmatrix}
= [A^{(2)} | b^{(2)}] = M_2 [A^{(2)} | b^{(2)}] = M_2 M_1 (A | b) =
\]

\[
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \ldots & a_{1n} & b_1 \\
0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots & a_{2n}^{(2)} & b_2 \\
0 & 0 & a_{33}^{(2)} & \ldots & a_{3n}^{(2)} & b_3 \\
0 & 0 & a_{43}^{(2)} & \ldots & a_{4n}^{(2)} & b_4 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & a_{n3}^{(2)} & \ldots & a_{nn}^{(2)} & b_n
\end{bmatrix}
\]

\[
= [A^{(3)} | b^{(3)}] =
\]

**Steps 3 to n-1**

If no \(a_{ij}^{(j)} = 0\), the process can continue and we get \(M_{n-1} M_{n-2} \ldots M_2 M_1 (A | b) = (U | y)\),

where \(U\) is of the form \((\mathbb{N})\).

Let \(L_1 = M_{n-1} M_{n-2} \ldots M_2 M_1\).

If we disregard the right-hand-side vector \(b\), we have

\[L_1 A = U ,\]

where \(U\) is upper triangular \(=(\mathbb{N})\).

**Definitions:**

A matrix \(U\) is upper triangular if \(u_{ij}^{(j)} = 0\) for \(i > j\).

A matrix \(L\) is lower triangular if \(l_{ij}^{(j)} = 0\) for \(i < j\).

All the \(M_i\) matrices, \(i = 1, 2, \ldots, n-1\), are lower triangular, which implies that their product \(L_1\) is lower triangular. Also, it can be shown that the inverse of a non-singular lower triangular matrix is lower triangular. If we denote \(L_1^{-1}\) by \(L\), then we have

\[A = L_1^{-1} U = L U .\]
Conclusion

What we do when we perform Gaussian elimination on a system of linear equations \( Ax = b \) is formally equivalent to an LU-decomposition of the matrix \( A \), i.e. a decomposition \( A = LU \), where \( L \) is lower and \( U \) is upper triangular.

Note that the right-hand-side vector is transformed in the same way as the matrix. We finally get the transformed right-hand-side \( y = L_1 b = L^{-1} b \).

Hence, with Gaussian elimination the system \( Ax = b \) is transformed into \( Ux = y \) by what is formally a premultiplication by \( L^{-1} \). The system \( Ux = y \) is upper triangular and can be solved by back-substitution.

\[ \ast \ast \ast \]

We shall now see what this straightforward elimination strategy can lead to when we, like a computer, work with a fixed number of digits.

Example 2

We have

\[
A = \begin{bmatrix} 0.001 & 1.200 & 2.000 \\ 1.207 & 2.051 & 1.963 \\ 1.006 & 2.002 & 3.000 \end{bmatrix}, \quad b = \begin{bmatrix} 3.201 \\ 5.221 \\ 6.008 \end{bmatrix}
\]

The calculations are now done with four decimal digits:

\[
\frac{a_{21}}{a_{11}} = 1207, \quad \frac{a_{31}}{a_{11}} = 1006
\]

\[
a_{22}^{(2)} = 2.051 - (1207 \times 1.200) = 2.051 - 1448 \approx -1446
\]

\[
a_{32}^{(2)} = 2.002 - (1006 \times 1.200) = 2.002 - 1207 = -1205,
\]

etc.

\[
M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1207 & 1 & 0 \\ -1006 & 0 & 1 \end{bmatrix}
\]

\[
M_1(A \ b) = \begin{bmatrix} 0.001 & 1.200 & 2.000 & 3.201 \\ 0 & -1446 & -2412 & -3859 \\ 0 & -1205 & -2009 & -3214 \end{bmatrix}
\]

Here all intermediate results, as well as the final new matrix elements, have been rounded to four decimal digits:

\[
\frac{a_{22}^{(2)}}{a_{22}^{(2)}} = -1205 \div -1446 \approx 0.8333
\]
\[ a_3^{(3)} = -2009 - 0.8333 \times (-2412) \approx -2009 + 2010 = 1 \]
\[ b_3^{(3)} = -3214 - 0.8333 \times (-3859) \approx -3214 + 3216 = 2. \]

Back-substitution gives

\[ x_3 = 2.000, \quad x_2 = -0.6674, \quad x_1 = 1.900, \]

that is,

\[ x = \begin{bmatrix} 1.900 \\ -0.6674 \\ 2.000 \end{bmatrix}. \]

But the exact solution is

\[ x = \begin{bmatrix} 1.000 \\ 1.000 \\ 1.000 \end{bmatrix}. \]

* * *

Can we do better than in Example 2 with the same set of equations? The answer is Yes, as will be shown in Example 3.

**Example 3**

This is the same set of equations as in Example 2, but with the rows in a different order. As in Example 2, we work with four decimal digits:

\[
(\mathbf{A}|b) = \begin{bmatrix} 1.207 & 2.051 & 1.963 & 5.221 \\ 0.001 & 1.200 & 2.000 & 3.201 \\ 1.006 & 2.002 & 3.000 & 6.008 \end{bmatrix},
\]

\[ \frac{a_{21}}{a_{11}} = 0.0008285, \quad \frac{a_{31}}{a_{11}} = 0.8335, \]

\[ M_1(\mathbf{A}|b) = \begin{bmatrix} 1.207 & 2.051 & 1.963 & 5.221 \\ 0 & 1.198 & 1.998 & 3.197 \\ 0 & 0.292 & 1.364 & 1.656 \end{bmatrix}, \]

\[ \frac{a_{32}^{(2)}}{a_{22}^{(2)}} = 0.2437, \]

\[ M_2M_1(\mathbf{A}|b) = \begin{bmatrix} 1.207 & 2.051 & 1.963 & 5.221 \\ 0 & 1.198 & 1.998 & 3.197 \\ 0 & 0 & 0.8771 & 0.8769 \end{bmatrix} \]

This gives

\[ x = \begin{bmatrix} 1.002 \\ 1.001 \\ 0.9998 \end{bmatrix}. \]
4. PIVOTING

If in the Gaussian elimination, as described in Section 3, an \( a_{jj}^{(i)} \) is exactly 0, the algorithm would not work. It turns out that a small \( a_{jj}^{(i)} \) will give rise to big rounding errors. This is what happened in Example 2. In general, an attempt should be made to try to get large values of \( a_{11}, a_{22}, \ldots \) by suitable reordering of equations and/or unknowns. This reordering is called pivoting. The importance of it is demonstrated in Example 3 above.

There are different pivoting strategies. The two most used are:

a) Complete pivoting, which means that a search is made for the biggest element (in absolute value) in the whole remaining matrix. To get this element in the right position, both rows and columns may have to be interchanged. A column interchange corresponds to a reordering of the unknowns. [For example, an interchange between columns 1 and 2 corresponds to a reordering \( (x_1, x_2, x_3, \ldots, x_n) \to (x_2, x_1, x_3, \ldots, x_n) \).]

b) Partial pivoting, which means that a search is made for the biggest element (in absolute value) in the first column of the remaining matrix. Then only the rows, not the columns, may have to be reordered.

It turns out that: in practice, partial pivoting works sufficiently well in almost all cases.

The advantage of partial pivoting over complete pivoting is, of course, that it is faster, since it requires less searching for pivots and less book-keeping.

In Example 3, the order of the equations was the order given by the partial pivoting strategy, and, as we have seen, this worked out satisfactorily.

4.1 Formalism of partial pivoting

If we perform Gaussian elimination with partial pivoting, a reordering of rows may be done between each elimination step; that is, between the premultiplication by \( M_i \) matrices. This reordering can be expressed as a premultiplication by permutation matrices. A permutation matrix is a matrix where each row and each column contains exactly one 1, and the rest of the elements are zeros. Premultiplication by a permutation matrix changes the order of the rows of a matrix, and postmultiplication changes the order of the columns.

So Gaussian elimination with partial (row) pivoting can formally be written as

\[
M_{n-1} P_{n-1} M_{n-2} P_{n-2} \ldots M_2 P_2 M_1 P_1 A = U ,
\]

where \( P_1, P_2, \ldots, P_{n-1} \) are permutation matrices. It can be shown [see for instance Wilkinson\(^1\)] that this is equal to

\[
\tilde{M}_{n-1} \tilde{M}_{n-2} \ldots \tilde{M}_2 \tilde{M}_1 \tilde{P} A = U ,
\]

where \( \tilde{M}_1, \tilde{M}_2, \ldots \) are matrices of the same type as \( M_1, M_2, \ldots \), and \( \tilde{P} \) is a permutation matrix. (Intuitively this is clear; if we had known the right order of the equations we could have started by performing the right reordering. This could be done with a permutation matrix \( \tilde{P} \).

As before, we get \( \tilde{P}A = LU \) (lower triangular \( \times \) upper triangular) or \( A = PLU \), where \( P = \tilde{P}^{-1} \) is also a permutation matrix.
Complete pivoting with interchanges of columns would correspond to a postmultiplication by a permutation matrix. Instead of the above decomposition \( A = PLU \), we get a decomposition \( A = P_1LUP_2 \).

Note

In practice, and for programming purposes, this formalism is not used. The \( M_i \) matrices are never formed, since only the numbers \( a_{21}/a_{11}, a_{31}/a_{11}, \) etc. are needed. These numbers can then be stored in the matrix in positions where the corresponding zeros are known to be produced. The rows do not have to be interchanged in the actual computer storage. Instead, an index vector can be used to indicate in which order the rows should be treated.

5. SCALING OF THE COEFFICIENT MATRIX

It is easy to construct examples where the partial pivoting strategy does not give very good results, simply because the problem is given in an unsuitable form. This will be shown in the following example.

Example 4

\[
A = \begin{bmatrix}
2.000 & 2400 & 4000 \\
1.207 & 2.051 & 1.963 \\
1.006 & 2.002 & 3.000
\end{bmatrix}, \quad b = \begin{bmatrix}
6402 \\
5.221 \\
6.008
\end{bmatrix}.
\]

We use partial pivoting, and, since \( a_{11} = 2 \) is the largest element in the first column, no interchange is done in the first step:

\[
\frac{a_{21}}{a_{11}} = 0.6035, \quad \frac{a_{31}}{a_{11}} = 0.5030
\]

\[
M_1(A|b) = \begin{bmatrix}
2.000 & 2400 & 4000 & 6402 \\
0 & -1446 & -2412 & -3859 \\
0 & -1205 & -2009 & -3214
\end{bmatrix}.
\]

No interchange is done in the second step, \( \frac{a_{32}}{a_{22}} = 0.8333 \):

\[
M_2M_1(A|b) = \begin{bmatrix}
2.000 & 2400 & 4000 & 6402 \\
0 & -1446 & -2412 & -3859 \\
0 & 0 & 1 & 2
\end{bmatrix},
\]

which gives

\[
x = \begin{bmatrix}
1.900 \\
-0.6674 \\
2.000
\end{bmatrix}.
\]

The correct answer is

\[
x = \begin{bmatrix}
1 \\
1 \\
1
\end{bmatrix}.
\]
As the reader may have noticed, Example 4 is the same as Example 2 but with the first equation multiplied by 2000.

* * *

Example 4 shows how important it is for this and similar methods that the problem be stated in such a way that the matrix is well balanced (or well scaled or equilibrated).

The problem of balancing (scaling, equilibrating) a matrix is not satisfactorily solved. It is normally favourable to have the elements of the matrix of roughly the same magnitude. Therefore, one commonly used strategy to improve a given matrix is the following: each row (an equation) in the matrix is multiplied, so that its biggest element, in absolute value, is 1. The corresponding element of the right-hand-side vector must also be multiplied by the same factor. Then the columns can be scaled the same way. If the kth column has been multiplied by αk, then the computed xk must be replaced by xk/αk.

If only row pivoting is done, scaling of the columns does not affect the solution process. However, if a program tests the size of a pivot element in order to detect ill-conditioning (which will be discussed in Sections 8 and 10.1 of this chapter), column scaling is of importance. Then, for partial pivoting the multiplications of the columns do not have to be performed, but the scale factors can be kept.

Scaling does not change the problem mathematically, but it affects the performance of numerical algorithms.

6. THE CROUT AND DOOLITTLE METHODS

Gaussian elimination without pivoting is equivalent to LU-decomposition. This decomposition, if it exists, is unique if we require k_{11} = 1 (which we have in the case of Gaussian elimination, as described in Section 3).

If we set

\[
\begin{bmatrix}
  a_{11} & a_{12} & \ldots & a_{1n} \\
  a_{21} & a_{22} & \ldots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \ldots & a_{nn}
\end{bmatrix}

= \begin{bmatrix} 1 & \xi_{21} & \xi_{31} & \xi_{32} & \cdots & \xi_{n1} & \xi_{n2} & \cdots & \xi_{n,n-1} & 1 \\
  \xi_{21} & 1 & \xi_{32} & \cdots & \xi_{n2} & \cdots & \xi_{n,n-1} & 1 \\
  \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
  \xi_{n1} & \xi_{n2} & \cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
  u_{11} & u_{12} & \cdots & u_{1n} \\
  u_{22} & \cdots & u_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  u_{n1} & u_{n2} & \cdots & u_{nn}
\end{bmatrix}

= LU,
\]

then we can determine the elements k_{ij} and u_{ij} directly just from the matrix multiplication rule. This gives what is called Doolittle's method.

If instead we require u_{11} = 1 and allow k_{11} ≠ 1, then again the decomposition, if it exists, is unique, and the L and the U matrix can be determined directly from the matrix multiplication rule. This is Crout's method. Formulae for Crout's method are given below:

\[
\begin{align*}
k_{ik} &= a_{ik} - \sum_{r=1}^{k-1} k_{ir} u_{rk} & \quad & i \leq k \\
u_{ik} &= \frac{1}{k_{ii}} \left( a_{ik} - \sum_{r=1}^{i-1} k_{ir} u_{rk} \right) & \quad & i > k,
\end{align*}
\]

where k = 1, ..., n, and, for each k, i = 1, ..., n.
After the decomposition is done, we solve $Lz = b$ and $Ux = z$.

Row pivoting can be included in the algorithms. Numerically, the methods work as well as Gaussian elimination. If sums of the type $\sum_{i=1}^{m} a_i b_i$ can be accumulated in double precision, Crout's or Doolittle's method has better numerical properties.

7. **CHOLESKY DECOMPOSITION FOR REAL SYMMETRIC, STRICTLY POSITIVE MATRICES**

We first need some notations and definitions (for completeness, these are given for the general complex case although only real matrices will be treated in the rest of this section):

- $A^T$ is the transpose of $A$.
- $A^* = \overline{A^T}$ is the complex conjugate transpose of $A$.

[If $a_{ij}$ is the $(i,j)$ element of $A$, $\overline{a_{ij}}$ is the $(j,i)$ element of $A^*$.]

Note that for any matrices $A$ and $B$, $(AB)^* = B^*A^*$.

**Definition:** A matrix $A$ is Hermitian if $A = A^*$.

A real Hermitian matrix is also symmetric, i.e. $A^T = A$ (since $A$ is real $A^* = A^T$).

The number $x^*Ax$ is real for all vectors $x$ when $A$ is Hermitian, since

$$x^*Ax = (x^*Ax)^* = x^*A^*x = x^*Ax.$$

**Definition:** If $A = A^*$ and $x^*Ax > 0$ for all vectors $x \neq 0$, then $A$ is (strictly) positive definite.

Some properties of a strictly positive matrix $A$:

i) $a_{ii} > 0$ for $i = 1, 2, \ldots, n$.

**Proof:** Choose $x$ as $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^T$, where the $1$ is in position $i$. Then $0 < e_i^T A e_i = a_{ii}$.

ii) All the eigenvalues of $A$ are positive.

**Proof:** Choose $x$ as an eigenvector. (For the definition of eigenvalue and eigenvector, see Chapter III.) $Ax = \lambda x \Rightarrow x^*Ax = \lambda x^*x$. Since $x^*x > 0$, we get $\lambda > 0$.

iii) $A$ is non-singular.

**Proof:** If $A$ is singular, then there is a vector $x \neq 0$ such that $Ax = 0$ and hence $x^*Ax = 0$.

---

Suppose $A$ is real and symmetric ($A = A^T$) and has an LU-decomposition

$$A = LU = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}$$

as defined in Section 3.
Then
\[ LU = A = A^T = (LU)^T = U^T L^T = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 0 & \cdots & 1 & 0 \end{bmatrix} \]

Can we obtain a similar factorization \( A = \tilde{U}\tilde{L} \), where \( \tilde{U} = \tilde{L}^T \)? We can try to let
\[
\begin{bmatrix}
    a_{11} & a_{21} & \cdots & a_{n1} \\
    a_{21} & a_{22} & \cdots & a_{n2} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix} =
\begin{bmatrix}
    \ell_{11} & 0 \\
    \ell_{21} & \ell_{22} \\
    \vdots & \ddots \\
    \ell_{n1} & \cdots & \ell_{nn}
\end{bmatrix}
\begin{bmatrix}
    k_{11} \\
    k_{21} \\
    \vdots \\
    k_{n1}
\end{bmatrix}
\]

Straightforward matrix multiplication gives
\[
\begin{align*}
    a_{11} &= \ell_{11}^2 \\
    a_{21} &= \ell_{11} \ell_{21} \\
    \vdots \\
    a_{n1} &= \ell_{11} \ell_{n1} \\
    a_{22} &= \ell_{21}^2 + \ell_{22}^2 \\
    a_{32} &= \ell_{31} \ell_{21} + \ell_{32} \ell_{22} \\
    \vdots \\
    a_{ij} &= \ell_{ij} \\
\end{align*}
\]

The general formulae are:
\[
\begin{align*}
    \ell_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2} \\
    \ell_{ji} &= \left( a_{ji} - \sum_{k=1}^{i-1} \ell_{jk} \ell_{ik} \right) / \ell_{ii}
\end{align*}
\]

where \( j \) can go from \( i + 1 \) to \( n \) for each \( i \), and \( i \) goes from \( 1 \) to \( n \).

(The computation can also be done in the order \( \ell_{11}, \ell_{21}, \ell_{22}, \ell_{31}, \ell_{32}, \ell_{33}, \ell_{41}, \ldots \).)

If any of the quantities of which the square root is required is negative, complex numbers would occur, and if any \( \ell_{ii} \) is zero the algorithm breaks down. It can be shown that if the matrix \( A \) is real, symmetric, and positive definite, this will not happen. The decomposition obtained is called the Cholesky decomposition.

**Number of multiplications**

\[
    \text{LU decomposition} = \frac{n^3}{3} \\
    \text{Cholesky decomposition} = \frac{n^3}{6} + n \text{ square roots}.
\]
The number of additions are (as often for matrix computations) approximately the same as the number of multiplications. The Cholesky decomposition takes advantage of the symmetry and thus brings down the number of multiplications by a factor of approximately 2 as compared with the LU decomposition. But then for the Cholesky decomposition as described, n square roots must be computed. The number of divisions is n for both methods.

Cholesky decomposition, and Gaussian elimination without pivoting (LU decomposition), work well for symmetric positive definite matrices.

8. ERROR ANALYSIS

It is important to distinguish between initially ill-conditioned problems and the computational difficulties which arise because a numerical method is unstable for a certain class of problems.

A given problem is ill-conditioned if a small perturbation of the input data may result in a big change in the solution. Suppose, for example, that a problem is defined by a set of input data, all of which have some small errors due to rounding, inaccuracies in measurements, etc. If then the possible solutions can vary greatly compared to the initial errors, the problem is ill-conditioned.

A method is unstable if it may produce a result which does not correspond to a problem close to the original one.

An attempt to visualize the different concepts is done in Figs. 1 and 2; P is a class of problems and S the set of corresponding solutions.

a) Well-conditioned problem

\[ P = \text{problems, } S = \text{solutions} \]

\[ P_1 \times P_2 \hspace{1cm} \times S_1 \hspace{1cm} \times S_2 \]

\[ P_1 \text{ and } P_2 \text{ close } \rightarrow S_1 \text{ and } S_2 \text{ close} \]

b) Ill-conditioned problem

\[ P \]

\[ P_1 \times P_2 \hspace{1cm} \times S_1 \hspace{1cm} \times S_2 \]

\[ P_1 \text{ and } P_2 \text{ close } \not\Rightarrow S_1 \text{ and } S_2 \text{ close} \]

Fig. 1 The condition of the problem

Comment on Fig. 2: The above presentation is somewhat simplified. (For instance, a stable method used on some problem \( P_1 \) in \( P \) might produce a "solution" which is not in \( S \).)

a) Stable method

\[ P \]

\[ P_1 \times P_2 \hspace{1cm} \times S_1 \hspace{1cm} \times S_2 \]

\[ P_1 \text{ and } P_2 \text{ close } \rightarrow S_1 \text{ and } S_2 \text{ close} \]

b) Unstable method

\[ P \]

\[ P_1 \times P_2 \hspace{1cm} \times S_1 \hspace{1cm} \times S_2 \]

For the problem \( P_1 \) the method gives the result \( S_2 \), which is the exact solution of a problem \( P_2 \) which is close to \( P_1 \).

For \( P_1 \) the method gives the result \( S_2 \). It is possible that there is no \( P_2 \) close to \( P_1 \) with \( S_2 \) as an exact solution.

Fig. 2 The stability of a numerical method
Obviously some metrics must be defined, so that it can be specified when, for example, two solutions are close. (Some vector and matrix norms will be introduced in Section 9.)

8.1 Backwards error analysis for numerical methods

If we try to estimate the error between the computed solution and the correct solution to a problem, we have to take both the conditioning of the problem and the stability of the method into account. However, this can be done in two steps, where the first one consists of giving estimates for the method, showing that a solution produced by it is the exact solution of a problem "close" to the original one.

For example, if we had computed an LU decomposition from a matrix A, we would assume that this was in fact a decomposition of A + E for some matrix E, and we would try to give some bounds on the elements of E.

Error analysis of this kind may be carried out for PLU decomposition, for the solution of a triangular system of linear equations (back substitution), and for other algorithms.

8.2 Some error bounds

Some upper bounds will be given for the errors introduced when Gaussian elimination is performed. The errors will be regarded as corresponding to a perturbation of the original coefficient matrix. It should be noted that these are upper bounds and that, in practice, the real errors are almost always much smaller.

If we compute the PLU decomposition of a matrix A, we get \( \text{PLU} = A + E \), where the elements \( e_{ij} \) of E satisfy

\[
|e_{ij}| < n \times \text{const.} \times \max |a_{ij}| \times 10^{-t} \times \gamma .
\]

Here \( 10^{-t} \) is the relative precision of the computer, \( n \) is the order of the matrix, and \( \gamma \) is a parameter which depends on the pivoting strategy and on the matrix. The parameter \( \gamma \) is defined as

\[
\gamma = \max_{k} \frac{\max_{i,j} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} .
\]

It can be shown that for positive definite matrices, \( \text{LU} \) decomposition gives \( \gamma = 1 \). For any matrix partial pivoting gives \( \gamma < 2^{n-1} \). Also, for any matrix and complete pivoting it can be proved that

\[
\gamma < \sqrt[n]{2 \times 3^{1} \times 4^{1} \ldots n^{1/n-1}} .
\]

We recall that complete pivoting corresponds to a \( P_{1}LJP_{2} \) decomposition where both \( P_{1} \) and \( P_{2} \) are permutation matrices. The question if \( \gamma \leq n \) can be shown for complete pivoting is still open.

If we use Crout's method (described in Section 6) and accumulate inner products in double precision, then the factor \( n \) can be removed from the bound (I.1). We get

\[
|e_{ij}| < \text{const.} \times \max |a_{ij}| \times 10^{-t} \times \gamma .
\]
If we solve $Ax = b$ by any of the decomposition methods described above, followed by back-substitution, we get $\tilde{x}$ such that $(A+F)\tilde{x} = b$ and

$$|f_{ij}| < (n \times c_1 + 2n^2c_2 + n^3c_3 \times 10^{-t}) \gamma \times 10^{-t} \times \max_{i,j} |a_{ij}|,$$

where $c_1$, $c_2$ are constants. (All constants appearing in the expressions above are of order unity.)

For more bounds and error analyses see, for example, Wilkinson\textsuperscript{1)} and Stewart\textsuperscript{2}).

We now have a solution for $A + E$ and some upper bounds on $E$. How much does the real solution change when we add this (unknown) perturbation matrix $E$ to $A$? This depends on the condition of the problem and this will be treated in the next section.

9. **ILL- AND WELL-CONDITIONED PROBLEMS; CONDITION NUMBER FOR INVERSION**

Here we will try to answer the following question: If we have $(A+E)\tilde{x} = b$, where $E$ is a small perturbation to the matrix $A$, and if $x$ denotes the correct solution of $Ax = b$, how much can $x$ and $\tilde{x}$ differ?

In order to answer this, we will need the concept of vector and matrix norms. We will assume that the reader is familiar with the general definition and the properties of norms \textsuperscript{[if not, see for example Forsythe and Moler\textsuperscript{3}) or Stewart\textsuperscript{2}] and only give some useful vector norms and corresponding matrix norms.

**Definition:** If we have a vector norm, the subordinate matrix norm is defined as

$$||A|| = \max_{x \neq 0} \frac{||Ax||}{||x||},$$

which is equivalent to

$$\max_{||x|| = 1} ||Ax||.$$

It follows that $||AB|| \leq ||A|| \cdot ||B||$ for any matrices $A$, $B$.

Some often used vector norms and their subordinate matrix norms are:

$$||x||_{\infty} = \max_{i} |x_i| \quad ||A||_{\infty} = \max_{i} \sum_{j=1}^{n} |a_{ij}|$$

$$||x||_{1} = \sum_{i=1}^{n} |x_i| \quad ||A||_{1} = \max_{j} \sum_{i=1}^{n} |a_{ij}|$$

$$||x||_{2} = \left( \sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}} \quad ||A||_{2} = \left[ \max_{i} \lambda_i(A^*A) \right]^{\frac{1}{2}},$$

where $\lambda_i(A^*A)$, $i = 1, \ldots, n$ are the eigenvalues of $A^*A$. 
For all these norms, the norm of the identity matrix \( I = \begin{pmatrix} 1 & \cdot & 0 \\ \cdot & \cdot & \cdot \\ 0 & \cdot & 1 \end{pmatrix} \) is 1, and for any non-
singular matrix \( A \) we get the inequality

\[
1 = \| I \| = \| AA^{-1} \| \leq \| A \| \| A^{-1} \| .
\]

Bound on the perturbation of the solution:

If \( Ax = b \) and \((A+E)(x+\Delta x) = b\), then it can be shown that

\[
\frac{\| \Delta x \|}{\| x \|} \leq \frac{\kappa(A) \| E \|}{1 - \kappa(A) \| A \|},
\]

(1.2)

where \( \kappa(A) = \| A \| \| A^{-1} \| \).

This gives a bound on the relative error in the solution, \( \| \Delta x \| / \| x \| \), in terms of the
relative perturbation of the matrix \( A \), \( \| E \| / \| A \| \), and of \( \kappa(A) \). This estimate holds for all
of the defined norms: \( \| \cdot \|_{\infty} \), \( \| \cdot \|_{1} \), and \( \| \cdot \|_{2} \).

**Definition:** \( \kappa(A) = \| A \| \| A^{-1} \| \) is called the condition number for inversion of \( A \)
(for the norm chosen).

Since \( \| A \| \| A^{-1} \| \geq \| AA^{-1} \| = 1 \), we have \( \kappa(A) \geq 1 \).

To be able to use the estimate (1.2) for any norm, we need the inverse of \( A \), \( A^{-1} \). But
\( A^{-1} \) has so far not been needed in the calculations, since we have performed some kind of
LU decomposition followed by back-substitution.

How expensive is it to calculate \( A^{-1} \)?

**LU decomposition = \( n^3/3 \) multiplications (already done)**

\[
U^{-1} = n^3/6 \quad "
\]

\[
L^{-1} = n^3/6 \quad "
\]

\( A^{-1} \) as \( U^{-1}L^{-1} = n^3/3 \quad "

The extra work will be of the order \( 2n^3/3 \) multiplications. The LU decomposition and the
back-substitution only takes \( n^3/3 \) multiplications. Hence the computation of \( A^{-1} \) is re-
latively costly and should be avoided! We therefore prefer not to use the estimate (1.2) directly.

10. **PRACTICAL WAYS OF DETECTING ILL-CONDITIONING**

10.1 **Test on pivots**

The size of the pivot elements can give some information as to whether or not the
matrix is ill-conditioned. The pivot elements are

\[
u_{11} (= a_{11}), \; u_{22} (= a_{22}^{(2)}), \; u_{33} (= a_{33}^{(3)}), \; \ldots, \; u_{n-1,n-1} (= a_{n-1,n-1}^{(n-1)}) .
\]

Include \( u_{nn} (= a_{nn}^{(n)}) \).
Let \( \det(A) \) denote the determinant of \( A \),

\[
\det(A) = \det(P) \times \det(L) \times \det(U) ,
\]

where \( U \) is upper triangular. Therefore \( \det(A) = \det(U) = u_{11} \times u_{22} \times \ldots \times u_{nn} . \)

Somewhat vaguely we can say that a small pivot element implies a small determinant, which shows that the problem is ill-conditioned. But, unfortunately, ill-conditioning does not always give rise to some small pivot elements. Therefore tests on the size of the pivots is not a sufficient safeguard against ill-conditioning. This is illustrated in the following example.

**Example 5**

\[
A = \begin{bmatrix}
1 & 1 & -1 & 1 & -1 & 1 & -1 \\
0.33 & 1 & -1 & 1 & -1 & 1 \\
0.33 & 1 & -1 & 1 & -1 & 0.33 \\
0.33 & 1 & -1 & 1 & 0.33 \\
0 & 1 & 1
\end{bmatrix}
\quad b = \begin{bmatrix} 1 \\ 1.33 \\ 0.33 \\ 0.33 \\ 1 \end{bmatrix}
\]

\( Ax = b \) has the exact solution \( x = (1, 1, 1, 1, 1, 1)^T \).

The matrix \( A \) is balanced, already upper triangular, and the pivot elements are 1 or 0.33.

We have no small pivot elements, but is the matrix well-conditioned? The answer is NO.

Perturb the elements that are 0.33 to 1/3. The perturbation is 1/300.

\[
A + E = \begin{bmatrix}
1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1/3 & 1 & -1 & 1 & -1 & 1 & -1 \\
1/3 & 1 & -1 & 1 & -1 & 1 & -1 \\
1/3 & 1 & -1 & 1 & 0.99 \\
0 & 1 & 1
\end{bmatrix}
\]

gives \( \tilde{x} = \begin{bmatrix} 3.05 \\ -0.54 \\ 1.38 \\ 0.90 \\ 1.02 \\ 0.99 \\ 1.00 \end{bmatrix} \)

for the same right-hand side \( b \); \( \tilde{x} \) is computed without any rounding errors.

** * * * **
10.2 **Iterative improvement**

A better way (which, however, requires some extra work) of detecting ill-conditioning is to perform one step of iterative improvement.

Suppose that we have a matrix $A$ and a right-hand-side vector $b$, and that we have computed a solution $x_1$ to $Ax_1 = b$. Because of rounding errors, $x_1 \neq x$. We then want to compute a correction $\Delta x_1$ to $x_1$ such that

$$x = x_1 + \Delta x_1 .$$

Then

$$0 = Ax - b = Ax_1 - b + A\Delta x_1 ,$$

and thus

$$A\Delta x_1 = -(Ax_1 - b) .$$

Put $r_1 = Ax_1 - b$. Then $\Delta x_1$ is the solution of $A\Delta x_1 = -r_1$.

Numerically, we proceed as follows:

i) Compute $r_1 = Ax_1 - b$ in double precision.

ii) Compute $\Delta x_1$ as the solution to $A\Delta x_1 = -r_1$ in single precision.

iii) Compute $x_2 = x_1 + \Delta x_1$.

iv) Compute the relative improvement $\|\Delta x_1\|/\|x_1\|$ in some suitable norm.

*Comments on (i)*: The computation of the residual $r_1$ must be done in double precision to be of any use. In single precision $Ax_1 = b$, and there would be a loss of significant digits when the subtraction $Ax_1 - b$ is performed.

*Comments on (ii) and (iii)*: Use the (normally) already computed decomposition of $A$ to solve this second system $A\Delta x_1 = -r_1$. Then only two back-substitutions have to be done. This takes $n^2$ multiplications. Owing to rounding errors, the computed $\Delta x_1$ will not be the exact solution. Therefore, the computed $x_2$ is not equal to $x$, the exact solution, but it is a new approximation to it. Obviously the whole process can be repeated with $x_2$ to get a new approximation $x_3$, etc.

*Comments on (iv)*: $x_1$ is the exact solution of $(A+E)x_1 = b$ for some small matrix $E$, which depends on the computational method. If the problem of solving $Ax = b$ is well-conditioned, then $x_1$ is a good approximation to $x$, and the computed $\Delta x_1$ a good approximation of the real solution of $A\Delta x_1 = -r_1$. Then

$$\frac{\|\Delta x_1\|}{\|x_1\|} = \frac{\|x_1 - x\|}{\|x\|} = \text{the relative error in } x_1 .$$

If $\|\Delta x_1\|/\|x_1\|$ is of the same order as the relative accuracy of the computer (for the CDC 7600 $\approx 10^{-14}$) then, for two reasons, it is not meaningful to make another step of iterative improvement. The first reason is that $Ax_2 = b$ in double precision, so the new residual
will have almost no significant digits. The second reason which often applies is that a
perturbation of the matrix A at machine rounding level will introduce an error in x of order
\( \Delta x_1 \); thus if the matrix is not represented exactly in the computer, we already have intro-
duced errors of this magnitude. When \( \| \Delta x_1 \| / \| x_1 \| \) is not at computer rounding level, but
still small, then the second argument still applies if the matrix is not exactly represented,
and again a second step of iterative improvement is not meaningful.

The conclusion of this is that more than one step of iterative improvement is only use-
ful when the matrix is exactly represented in the computer, and then only as long as the re-
sidual is computed with some significant digits. The number \( \| \Delta x_1 \| / \| x_1 \| \) is approximately
the relative error.

11. ITERATIVE METHODS

All methods discussed in the previous sections (Gaussian elimination, Crout's and
Doolittle's methods, Cholesky decomposition) are direct methods. They would, if exact arith-
metic could be performed, produce an exact solution of \( Ax = b \) in a fixed predictable number
of operations.

Systems of the type \( Ax = b \) can also be solved by iterative methods. The basic idea of
many iterative methods is the following:

Rewrite the equations on the form

\[ x = Mx + y, \]

where \( M \) is a known matrix and \( y \) a known vector, and \( x \) still denotes the wanted solution.
Then choose a starting vector \( x^{(0)} \) and use the formula

\[ x^{(m+1)} = Mx^{(m)} + y, \quad m = 0, 1, \ldots. \]

Iterations of this type converge under certain conditions on the matrix \( M \). There are many
ways to rewrite the system \( Ax = b \) on the form \( x = Mx + y \), each giving a different iteration
formula.

Iterative methods are very seldom used for small dense systems, i.e. systems where the
full coefficient matrix can be stored in the computer and where the coefficient matrix does
not contain a large number of zeros. For large sparse systems, however, the situation is
different. A sparse matrix is a matrix which contains a big percentage of zeros.

A matrix which is too large to be stored in the fast memory of a computer can still be
handled efficiently if it is a sparse matrix, since then only the non-zero elements need to
be stored (together with information about where in the matrix these non-zero elements are
located).

When direct methods, for example Gaussian elimination, are used on a system with a sparse
coefficient matrix, the transformations done on the matrix normally increase the number of
non-zero elements considerably. Therefore, iterative methods are often applied to large
sparse systems.
If the coefficient matrix has any special structure or special properties, this should be taken into account when an iterative scheme is chosen.

If the coefficient matrix is a band matrix, i.e. if \( a_{ij} = 0 \) when \(|i - j| > m\) for some \( m << n \) (where \( n \) is the order of the matrix), then a direct method might still be preferable even if the matrix is large. Gaussian elimination with row pivoting can be performed and stored in a band matrix with \( m \) additional bands.

Iterative methods and sparse matrices will not be discussed further in this report, but some recent books in these fields are listed after the other references.
CHAPTER II

LINEAR LEAST SQUARES PROBLEMS

The problem of finding the best solution, in the least squares sense, to an over-determined system of linear equations will be discussed.

Householder transformations will be introduced, and the method using this technique to solve the linear least squares problem by transforming the matrix to upper triangular form will be presented. A brief introduction to the method of singular value decomposition will also be given.

1. HOUSEHOLDER TRANSFORMATIONS

We will need some prerequisites for the rest of this Chapter.

Notation: If \( A \) is a matrix, then let \( a_i \) denote the \( i \)th column of \( A \). Hence
\[
A = (a_1, a_2, ..., a_n).
\]

Definition: If for an \( n \times n \) matrix \( U \) the relation \( U^* U = U U^* \) holds, the matrix is called a unitary matrix.

In a unitary matrix, the columns are normalized and orthogonal to each other. That is, if \( U = (u_1, u_2, ..., u_n) \) is unitary, then
\[
u_i^* u_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}
\]
The corresponding relations hold for the rows of a unitary matrix. If a unitary matrix is real, it is also called orthogonal.

Let \( U \) be a unitary matrix and \( x \) an arbitrary vector. Then \( \|Ux\|_2 = (Ux)^* (Ux) = x^* U^* U x = x^* x = \|x\|_2^2 \).

Since \( \|U\|_2 = \max_{x \neq 0} (\|Ux\|_2 / \|x\|_2) \), it follows directly that \( \|U\|_2 = 1 \).

It may also be shown that for any \( n \times n \) matrix \( A \), \( \|UA\|_2 = \|A\|_2 \). Hence a unitary transformation of a vector or a matrix always preserves the norm \( \|\cdot\|_2 \).

Since unitary transformations are norm preserving, they have very good numerical properties. They do not amplify errors that are already introduced. Suppose that instead of the matrix \( A \) we have the matrix \( A + \epsilon \), where \( \epsilon \) is a small perturbation. Premultiplication (in exact arithmetic) by a unitary matrix \( U \) then gives, instead of \( UA = B \), the matrix \( U(A + \epsilon) = UA + UE = B + F \). Here \( \|F\| = \|E\| \), so the perturbation remains of the same magnitude.
We will now introduce a special class of unitary transformation, the Householder transformation.

**Definition:** Let \( x \) be a non-zero vector, normalized such that \( \|x\|_2^2 = x^*x = \sum_{i=1}^{n} |x_i|^2 = 1 \), and let \( I \), as before, denote the identity matrix. Then the matrix

\[
H = I - 2xx^*
\]

is a Householder matrix or an elementary reflector matrix, and a transformation of the form \( u = Hv \) is called a Householder transformation.

It is easy to show that a Householder matrix is both unitary and Hermitian; that is,

\[
H^{-1} = H^* = H.
\]

**Theorem:** If \( u \) and \( v \) are two vectors such that

i) \( \|u\|_2 = \|v\|_2 \),

ii) \( u^*v \) is real,

then there exists a Householder matrix \( H \) such that \( u = Hv \). [Note that (ii) is always true for real \( u \) and \( v \).]

**Proof:** Take \( x = \frac{v - u}{\|v - u\|_2} \).

\[
Hv = \left[ I - 2 \frac{(v - u)(v - u)^*}{\|v - u\|_2^2} \right] v = v - 2 \frac{(v - u)(v - u)^*v}{(v - u)^*(v - u)}.
\]

Using \( u^*u = v^*v \) from (i) and \( u^*v = v^*u \) from (ii), we obtain

\[
Hv = v - 2 \frac{v^*v - u^*v}{v^*v - v^*u - u^*v + u^*u} (v - u) = v - (v - u) = u.
\]

Q.E.D.

**Note:** In practice, the matrix \( H \) of a Householder transformation is not computed, since it is completely determined in terms of \( u \) and \( v \).

Householder transformations can be used to introduce zeros in matrices, since they can transform a column

\[
\begin{bmatrix}
a_{1v} \\ a_{2v} \\ \vdots \\ a_{nv}
\end{bmatrix}
\]

into

\[
\begin{bmatrix}
c \\ 0 \\ \vdots \\ 0
\end{bmatrix}
\]

where \( c = \pm \left( \sum_{i=1}^{n} |a_{iv}|^2 \right)^{1/2} = \pm \|a_v\|_2 \).

The sign is chosen such that \( a_{1v} \cdot c \leq 0 \) in order to avoid numerical cancellation (see below).

Normally, when the transformed vector \( u \) contains zeros in all positions except the first, only the original vector \( v \) is stored completely.
The computational implementation can then be done as follows:

Suppose we want the Householder transformation which takes

\[
\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad \text{into} \quad \mathbf{u} = \begin{bmatrix} \pm \|\mathbf{v}\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\]

Let

\[
\mathbf{t} = \mathbf{v} - \mathbf{u} = \begin{bmatrix} v_1 - \|\mathbf{v}\|_2 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}
\]

We get

\[
\|\mathbf{v} - \mathbf{u}\|_2^2 = \|\mathbf{v}\|_2^2 = 2v_1 \|\mathbf{v}\|_2 + \|\mathbf{u}\|_2^2 = 2(\|\mathbf{v}\|_2^2 + v_1 \|\mathbf{v}\|_2)
\]

Let \( k = \|\mathbf{v}\|_2 + v_1 \), where the sign should be taken as the sign of \( v_1 \) in order to avoid loss of accuracy by the cancellation of digits.

The transformation is then defined by the matrix

\[
\mathbf{H} = \mathbf{I} - \frac{2}{\|\mathbf{v} - \mathbf{u}\|_2^2} (\mathbf{v} - \mathbf{u})(\mathbf{v} - \mathbf{u})^* = \mathbf{I} - \frac{\mathbf{t}\mathbf{t}^*}{k}
\]

Thus the transformation is completely determined by one vector \( \mathbf{t} \) and the scalar \( k \).

If a matrix \( \mathbf{A} \) is premultiplied by \( \mathbf{H} \), the result will be

\[
\mathbf{HA} = \mathbf{H}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n) = \mathbf{A} - \frac{\mathbf{t}\mathbf{t}^*}{k} (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n) = \mathbf{A} - (\mathbf{t}^* \mathbf{a}_1, \mathbf{t}^* \mathbf{a}_2, \ldots, \mathbf{t}^* \mathbf{a}_n) \cdot \mathbf{t}
\]

Note that \((\mathbf{t}\mathbf{t}^*/k)\mathbf{a}_i = (\mathbf{t}^* \mathbf{a}_i/k) \mathbf{t}\), where \( \mathbf{t}^* \mathbf{a}_i/k \) is a scalar.

If the purpose of the transformation is to introduce zeros in a matrix, then \( v \) will be a column (or part of a column) of the matrix. This column then, except for the first element, already contains the vector \( \mathbf{t} \), and therefore only two additional numbers, \( t_1 \) and \( k \), are needed in order to define the transformation. Furthermore, if in a computer program the transformation has to be kept, this column, except for the first element, can be left unchanged. The transformation is known to produce zeros in these positions, and the zeros need not be stored.

2. THE LINEAR LEAST SQUARES PROBLEM

Let \( \mathbf{A} \) be an \( m \times n \) matrix, where \( m > n \), and \( \mathbf{b} \) a vector with \( m \) components:
\[
A = \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{bmatrix}, \quad
b = \begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_m
\end{bmatrix}
\]

We want a vector \( x \) with \( n \) components such that \( \|Ax - b\|_2 \) is minimized.

Since there are more equations than unknowns (the system \( Ax = b \) is overdetermined), we cannot, in general, expect to get \( Ax - b \) equal to 0.

Define \( r = Ax - b \), the residual vector. We want to minimize the norm of the residual

\[
\|Ax - b\|_2 = \|r\|_2 = \left[ \sum_{i=1}^{m} r_i^2 \right]^{1/2}
\]

which is the same as minimizing \( \sum_{i=1}^{m} r_i^2 \) (hence the name least squares).

Another way of looking at the problem is to say that we want to find that vector in the subspace spanned by the columns of \( A \), which is closest to \( b \) in the norm \( \| \cdot \|_2 \).

If the columns of \( A \) are linearly independent (no column can be expressed as a linear combination of the others) then there is a unique solution.

If not, then different combinations of the columns of \( A \) can give rise to the same vector, and the solution \( x \) is therefore not unique. There are then infinitely many combinations \( x \) which make \( \|Ax - b\|_2 \) a minimum.

When the columns of \( A \) are linearly independent, then \( A \) has rank \( n \), full rank.

A solution \( x \) to the least squares problem also satisfies the normal equations

\[
A^*Ax = A^*b.
\]

(Note that \( A^*A \) is an \( n \times n \) matrix and \( A^*b \) an \( n \times 1 \) vector.)

The normal equations are often very ill-conditioned.

Because of this, the solution can be very sensitive to perturbations of the matrix \( A^*A \), and big errors in the solution may already be introduced by the rounding errors when \( A^*A \) is computed.

3. THE QU-DECOMPOSITION

Any \( m \times n \) matrix \( A, m \geq n \), can be decomposed as \( A = QU \), where \( Q \) is a unitary \( m \times m \) matrix and the \( m \times n \) matrix \( U \) has the form

\[
U = \begin{pmatrix}
  U_1 \\
  0
\end{pmatrix}
\]

with \( U_1 \ n \times n \) upper triangular.
This can be done with Householder transformations in the following way:

**Step 1**

Choose $H_1$ such that

\[
\begin{bmatrix}
a_{11} \\
a_{21} \\
\vdots \\
a_{m1}
\end{bmatrix}
\] is transformed into

\[
\begin{bmatrix}
u_{11} \\
0 \\
\vdots \\
0
\end{bmatrix}
\]

\[
H_1A = \begin{bmatrix}
u_{11} & u_{12} & \ldots & u_{1n} \\
0 & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
0 & a_{m2} & \ldots & a_{mn}
\end{bmatrix}; \quad H_1 \text{ is } m \times m.
\]

**Step 2**

Choose $H_2$ such that

\[
\begin{bmatrix}
a_{22}^{(2)} \\
\vdots \\
a_{m2}^{(2)}
\end{bmatrix}
\]

is transformed into

\[
\begin{bmatrix}
u_{22} \\
0 \\
\vdots \\
0
\end{bmatrix}
\]

\[
H_2 = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & \tilde{h}_2 & \ddots & \vdots \\
\vdots & \vdots & \ddots & 0 \\
0 & 0 & \cdots & \tilde{h}_2
\end{bmatrix}, \quad H_2H_1A = \begin{bmatrix}
u_{11} & u_{12} & u_{13} & \ldots & u_{1n} \\
0 & u_{22} & u_{23} & \ldots & u_{2n} \\
0 & 0 & a_{33} & \ldots & a_{3n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & a_{m3} & \ldots & a_{mn}
\end{bmatrix}.
\]
Steps 3 to n

Steps 3 to n are analogous. After n steps we get

\[
H_n H_{n-1} \cdots H_2 H_1 A = \begin{bmatrix}
  u_{11} & u_{12} & \cdots & u_{1n} \\
  0 & u_{22} & \cdots & u_{2n} \\
  0 & 0 & \ddots & \vdots \\
  \vdots & \ddots & \ddots & u_{nn} \\
  0 & 0 & \cdots & 0
\end{bmatrix} \begin{bmatrix}
  U_1 \\
  0 \\
  \vdots \\
  \vdots \\
  0
\end{bmatrix} = U.
\]

Let \( Q^* = H_n H_{n-1} \cdots H_1 \).

Since \( H_1, H_2, \ldots, H_n \) are unitary, so is \( Q^* \), and therefore \( Q \).

When \( A \) has rank \( n \), the \( n \times n \) matrix \( U_1 \) will be non-singular.

**Comment:** This decomposition can be done also when rank \( A \) is \( < n \). In this case, take \( H_0 = \) the identity matrix when a zero column is encountered.

Let \( q_1, q_2, \ldots, q_m \) denote the columns of the \( m \times m \) matrix \( Q = (Q = H_0^* H_1^* \cdots H_n^* = H_n H_{n-1} \cdots H_1) \). We have \( A = U Q = (q_1, \ldots, q_n, q_{n+1}, \ldots, q_m) (U_1) = (q_1, \ldots, q_n) U_1 \) because of the zeros in \( U \).

Let \( Q_1 = (q_1, \ldots, q_n) \) and \( Q_2 = (q_{n+1}, \ldots, q_m) \). Then \( A = U Q = Q_1 U_1 \).

4. **SOLVING THE LINEAR LEAST SQUARES PROBLEM BY QU FACTORIZATION**

We want to minimize \( \| A x - b \|_2 \). Let \( A \) have the decomposition \( A = Q U_1 \) as described in the previous section. Since \( Q^* \) is unitary,

\[
\| A x - b \|_2 = \| Q^* A x - Q^* b \|_2 = \| U x - Q^* b \|_2 = \left\| \begin{bmatrix}
  U_1 x \\
  Q^*_2 b
\end{bmatrix} \right\|_2 = \left\| U_1 x \right\|_2 + \left\| Q^*_2 b \right\|_2.
\]

Let \( c_1 = Q^*_2 b, c_2 = Q^*_2 b \). Then \( c_1 \) is the vector containing the first \( n \) elements of \( Q^* b \).

Hence the solution \( x \) to our problem is the vector that minimizes

\[
\left\| \begin{bmatrix}
  U_1 x \ - c_1 \\
  - c_2
\end{bmatrix} \right\|_2.
\]

Now the last \( (m - n) \) components \( -c_2 \) do not depend on \( x \), so these elements cannot be decreased. The first \( n \) components can be set to zero if we choose \( x = U_1^{-1} c_1 \) and obviously this is the best we can do. Therefore

\[
\min \| A x - b \|_2 = \left\| \begin{bmatrix}
  0 \\
  -c_2
\end{bmatrix} \right\|_2 = \| c_2 \|_2 = \| Q^*_2 b \|_2 \quad \text{for} \quad x = U_1^{-1} Q^*_2 b.
\]
Computational algorithm

1) Compute \( Q^*A \) and \( Q^*b \) at the same time by Householder transformations. (Compute \( H_1A \), \( H_1b \), then \( H_2(H_1A) \), \( H_2(H_1b) \), etc. The matrices \( H_1, \ldots, H_n \) are not explicitly needed.)

2) After step 1, we have got \( A \rightarrow \begin{bmatrix} U_1^* \\ 0 \end{bmatrix}, \quad b \rightarrow \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \).

Solve \( U_1x = c_1 \) by back-substitution.

The error \( ||Ax - b||_2 \) can be computed as \( ||c_2||_2 \).

Optional: 3) If the so-called error matrix \( (A^*A)^{-1} \) is wanted, it can be computed as \( U_{n}^{-1}(U_{n}^{*})^* \) since \( A^*A = U_{n}Q_2Q_1U_1 = U_{n}U_{1} \).

The QU factorization requires \( \approx mn^2 - (n^3/3) \) multiplications. Computing \( A^*A \), the \( m \times m \) coefficient matrix of the normal equations, requires \( \approx mn^2/2 \) multiplications.

Then why use Householder transformations when it is faster to solve the normal equations?

The main reason is that the QU factorization method is more stable. Also, as described in the next section, with the QU factorization method the process can easily be stopped at an intermediate stage if it turns out that a combination of fewer than \( n \) columns of \( A \) gives an approximation which is sufficiently accurate.

5. OBTAINING A PRESCRIBED ACCURACY

Suppose we do not want the best solution, but only a solution which gives a sufficiently small residual.

After \( k \) steps in the factorization we have

\[
\begin{bmatrix}
H_k & H_{k-1} & \ldots & H_1 \\
0 & & & \\
\end{bmatrix} = \begin{bmatrix} U^{(k)} \\ 0 \end{bmatrix} \gamma^{(k)} = \begin{bmatrix} U^{(k)} \\ \gamma^{(k)} \end{bmatrix},
\]

where \( U^{(k)} \) is a \( k \times k \) upper triangular matrix and

\[
H_k \begin{bmatrix} g^{(1)} \\ g^{(2)} \end{bmatrix} = \begin{bmatrix} g^{(1)} \\ g^{(2)} \end{bmatrix}, \quad \text{k rows,} \quad m-k \text{ rows}
\]

Suppose we find that \( ||g^{(2)}||_2 \) is small enough. Then we can stop the Householder transformation process and approximate \( b \) by

\[
(a_1, a_2, \ldots, a_k, x) = (a_1, a_2, \ldots, a_k, a_{k+1}, \ldots, a_m)^{T},
\]

where \( x \) is the solution of \( U^{(k)}x = g^{(1)} \).

Then \( ||A^{T}x - b||_2 = ||g^{(2)}||_2 \).
We have then used the first k columns to obtain the required accuracy in the approximation. It is possible, however, that there exists a subset of fewer than k columns which gives the same accuracy.

At each step we can pick out the column which gives the most improvement to the approximation. Let us look at the first step: \( A = (a_1, a_2, \ldots, a_n) \). We can regard \( x = (0, 0, \ldots, 0)^T \) as a starting approximation which would give the residual \( r = -b \). We now want to choose one column, say \( a_v \), to approximate b. Then it can be shown that the norm of the new residual will be \( \|b\|_2 - \frac{|a_v^*b|}{\|a_v\|_2} \). Therefore, choose \( a_v \) as the column which maximizes \( \frac{|a_v^*b|}{\|a_v\|_2} \). Make \( a_v \) the first column of the matrix and perform step 1 in the QR factorization. (The columns do not have to be interchanged if an index vector is kept.)

Step 2 operates only on rows 2, 3, \ldots, m of the new matrix and on the new right-hand side. In this step, choose that vector

\[
a_v^{(2)} = \begin{bmatrix}
a_v^{(2)} \\
\vdots \\
a_m^{(2)}
\end{bmatrix}
\]

which maximizes

\[
\frac{|a_v^{(2)*}b^{(2)}|}{\|a_v^{(2)}\|_2}
\]

where \( b^{(2)} = \begin{bmatrix} b_2^{(2)} \\
\vdots \\
b_m^{(2)} \end{bmatrix} \)

The choice of column at each step is based on the current submatrix and the current transformed right-hand side.

Note that for this procedure it is not necessary that \( A \) has full rank. If a zero column is encountered, that whole column in the current submatrix can simply be disregarded, since it is then a linear combination of previously treated columns.

This method is described by Golub\(^4\).

6. THE CONDITION OF THE LEAST SQUARES PROBLEM

Let \( A \) be an \( m \times n \) matrix, \( m > n \), of rank \( n \). A condition number corresponding to the condition number for inversion of square non-singular matrices can then be defined.

Definition: \( \kappa(A) = [\kappa(A^*A)]^{1/2} \),

where \( \kappa(A^*A) = \|A^*A\| \cdot \|(A^*A)^{-1}\| \).

It can be shown that

i) if the residual is big then the "condition number" for the least squares problem is of order \( [\kappa(A)]^2 \). That is, a perturbation of order \( e \) will give rise to errors in the solution of order \( e[\kappa(A)]^2 \);
ii) if the residual is small, the "condition number" for the least squares problem is \( \approx \kappa(A) \).

The least squares problem can be solved by QU factorization as described above with an error growth proportional to the condition number of the problem.

**Warning:** If the normal equations \( A^*Ax = A^*b \) are formed, the original problem is transformed into another problem with the condition number \( \kappa(A^*A) = [\kappa(A)]^2 \).

Note that \( \kappa(A) \) is often very big!

7. **OTHER METHODS**

7.1 **Gram-Schmidt orthogonalization**

This corresponds to the use of orthogonal functions (for instance, orthogonal polynomials). The columns of the \( m \times n \) matrix \( A \) are orthogonalized. This can be done by a series of post-multiplications by upper triangular matrices, thus producing a matrix \( U_1 \ U_2 \ldots \ U_{n-1} \) with orthogonal columns, which can be written as \( Q_1 \ D \) where the columns of \( Q_1 \) are normalized and \( D \) is diagonal. Writing \( U = U_1 \ U_2 \ldots \ U_{n-1} \) we have

\[
A U = Q_1 \ D, \quad \text{or} \quad A = Q_1 \ U U^{-1} = Q_1 \ U,
\]

where \( U \) is upper triangular. Hence we have again produced a QU factorization.

The classical Gram-Schmidt method takes column \( k, k = 2, 3, \ldots, n \), and makes it orthogonal to all the previous columns. There is also a modified Gram-Schmidt method, which takes column \( k, k = 1, 2, \ldots, n-1 \), and makes all the following columns orthogonal to it.

Unfortunately, the method which corresponds to the classical Gram-Schmidt orthogonalization is numerically unstable. However, it turns out that the modified Gram-Schmidt process can be applied to the least squares problem, in spite of the fact that this method in other applications may also cause numerical problems.

For a detailed description, see for instance, Dahlquist and Björck\(^5\).

7.2 **The singular value decomposition and the Moore-Penrose generalized inverse**

Any \( m \times n \) matrix \( A \) can be decomposed as:

\[
A = U \left( \begin{array}{c}
\Sigma \\
0
\end{array} \right) V^* \quad \text{(the singular value decomposition)}.
\]

\( U \) and \( V \) are unitary matrices of order \( m \) and \( n \), respectively, and

\[
\Sigma = \begin{bmatrix} 
\sigma_1 & 0 \\
0 & 0 
\end{bmatrix}, \quad \text{where} \Sigma_1 = \begin{bmatrix} \sigma_1 & 0 \\
0 & \sigma_2 \\
& \ddots \\
0 & \sigma_r 
\end{bmatrix}.
\]

\( r \) is the rank of \( A \), and \( \sigma_1, \sigma_2, \ldots, \sigma_r \) are the non-zero singular values of \( A \); \( n-r \) singular values \( \sigma_{r+1}, \ldots, \sigma_n \) are zero.
It is easy to show that $\sigma_i^2$, $i = 1, \ldots, n$, must be the eigenvalues of $A^*A$; $\sigma_i$, $i = 1, \ldots, n$ are chosen to be non-negative.

If $r = n$ then $\Sigma = \Sigma_1$, and if $m < n$ the middle matrix is of the form $(\Sigma \ 0)$.

Since the singular value decomposition involves computation of eigenvalues and eigenvectors, the presentation of the decomposition method will be given in Chapter III, Section 5.

The singular value decomposition can be used for the solution of least squares problems. If the rank of the $m \times n$ ($m \geq n$) matrix $A$ is less than $n$, then there are infinitely many solution vectors $x$ which minimize $\|Ax - b\|_2$. In this case a zero column will be encountered in the QR factorization, and $U_1$ will be singular. We could, as mentioned before, just disregard the zero column and thus obtain one possible solution. However, among the infinite set of solution vectors there is one, say $\tilde{x}$, with the smallest norm. This vector can be computed with the singular value decomposition.

Once we have got the decomposition of $A$, the vector $\tilde{x}$, defined as the smallest vector which minimizes $\|Ax - b\|_2$, can be computed as

$$\tilde{x} = V \begin{bmatrix} \Sigma_1^{-1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^* b.$$  

The matrix

$$V \begin{bmatrix} \Sigma_1^{-1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^*$$

is the Moore-Penrose generalized inverse of $A$. If the rank of $A$ is $n$, then this is equal to $(A^*A)^{-1}A^*$.

$$\begin{bmatrix} 1/\sigma_1 \\ 1/\sigma_2 \\ \vdots \\ 1/\sigma_r \end{bmatrix}$$

We have $\Sigma_1^{-1}$

Note that the Moore-Penrose generalized inverse, and consequently $\tilde{x}$, are not continuous functions of the elements of the matrix $A$. When $\sigma_j$ is small but not zero, $\Sigma_1^{-1}$ will contain the element $1/\sigma_j$, but when $\sigma_j = 0$ this term will be replaced by 0. We therefore have the problem to decide when a computed $\sigma_j$ should be regarded as effectively zero.

The singular value decomposition also has applications in statistics.
CHAPTER III

THE EIGENVALUE PROBLEM $Ax = \lambda x$

Some mathematical background will be given for the eigenvalue problem $Ax = \lambda x$. The conditioning of the problem, i.e. the sensitivity of the eigenvalues and eigenvectors to small perturbations in the matrix $A$, will be discussed. The most important method for calculating eigenvalues, the QR method, will be described, as also the inverse iteration method for eigenvector calculations. Finally, as a complement to the previous chapter, a practical way of finding the singular value decomposition of a matrix will be outlined.

1. SOME MATHEMATICAL BACKGROUND

Let $A$ be an $n \times n$ matrix. We want vectors $x_1 \neq 0$ and scalars $\lambda_1$ such that $Ax_1 = \lambda_1 x_1$.

$Ax = \lambda x$ if and only if $(A - \lambda I)x = 0$, where $I$ is the identity matrix. But for $(A - \lambda I)x = 0$ to have a solution other than $x = 0$, it is necessary that $\det(A - \lambda I) = 0$.

The determinant of $A - \lambda I$, $\det(A - \lambda I)$, is a polynomial in $\lambda$ of degree $n$. It is called the characteristic polynomial of $A$. The $n$ roots $\lambda_1$, $\lambda_2$, ..., $\lambda_n$ are the eigenvalues of $A$, and the corresponding vectors $x_1$ are the eigenvectors. If $x_1 \neq 0$ is an eigenvector, i.e. $Ax_1 = \lambda_1 x_1$, then $kx_1$, where $k$ is a non-zero scalar, is also an eigenvector; $x_1$ and $kx_1$ are considered as the same eigenvector.

In the general case, the characteristic polynomial should not be computed. There are no direct methods for solving the eigenvalue problem, only iterative methods. The most important one is the QR method, which will be described in a later section.

Before computing, a check should be made to see if the matrix has any relevant special properties, so that we know what results to expect.

i) Real symmetric matrix: The eigenvalues are real and there are $n$ real orthogonal eigenvectors. If, for example, $\lambda_1 = \lambda_2$ (this eigenvalue then has multiplicity 2), two orthogonal eigenvectors $x_1$ and $x_2$ can be found, but they are not unique. Any linear combination $\alpha_1 x_1 + \alpha_2 x_2$ is an eigenvector.

ii) Real general matrix: The eigenvalues and eigenvectors are either real or occur in complex conjugate pairs. For an eigenvalue with multiplicity $m > 1$ there can be fewer than $m$ eigenvectors, as in the following example:

Example 1

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \lambda_1 = \lambda_2 = 1.$$
For an eigenvector \( \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \) we get the equations

\[
\begin{align*}
1 \times x_1 + 1 \times x_2 &= 1 \times x_1 \\
1 \times x_2 &= 1 \times x_2
\end{align*}
\]

which gives \( x_2 = 0 \)

Hence we only get one eigenvector of the form \( k \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} \)

iii) Complex Hermitian matrix: \( (C = C^*) \): The eigenvalues are real and there are \( n \) orthogonal eigenvectors which may be complex. [For multiple eigenvalues, as case (i).]

iv) Complex general matrix: Both the eigenvalues and eigenvectors may be complex, and an eigenvalue of multiplicity \( m \), \( m > 1 \) may have fewer than \( m \) eigenvectors.

A few more facts about eigenvalues and eigenvectors will be given below:

\( A^T \) and \( A \) have the same eigenvalues (but, in general, not the same eigenvectors).

A triangular matrix has the eigenvalues as diagonal elements.

\[
\prod_{i=1}^{n} \lambda_i = \det(A).
\]

\[
\sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} a_{ii} \text{ is the trace of } A.
\]

\[
\max |\lambda_i| \leq ||A||_2 \text{ since } ||A||_2 = \max_{x \neq 0} \frac{||Ax||_2}{||x||_2} \geq \frac{||Ax_1||_2}{||x_1||_2} = |\lambda_1|
\]

An Hermitian (e.g. real symmetric) positive definite matrix has positive eigenvalues.

* * *

The eigenvectors show which directions remain the same when the transformation \( A \) is applied. The corresponding eigenvalues show how much a vector in this direction is "stretched" (or "contracted"). Some other vectors may be "stretched" much more, but then the direction is changed. The maximum "stretch" is \( ||A||_2 = \sqrt{\lambda_{\text{max}}(A^*A)} = \) the largest singular value \( \sigma_1 \) of \( A \). The minimum "stretch" is \( \min \sigma_1 = \) the smallest singular value. When \( A = A^* \) then \( \max |\lambda_i| = \max \sigma_i = ||A||_2 = \max \sigma_1 \).

Every \( n \times n \) matrix \( A \) can be decomposed as \( A = UH \), where \( U \) is unitary and \( H \) is Hermitian with the singular values of \( A, \sigma_1, \sigma_2, \ldots, \sigma_n \) as eigenvalues. Hence every matrix transformation can be thought of as first a "stretch" in \( n \) orthogonal directions and then a rotation.
2. **The Conditioning of the Eigenvalue Problem**

To start with, let us consider a given method for computing eigenvalues and eigenvectors as a "black box" and discuss the interpretation of the results.

Suppose we have used a method which is known to be stable, in the sense that it produces the eigenvalues and eigenvectors of $A + E$, where $E$ is some small perturbation, instead of those of our matrix $A$. We would then want to know how much this perturbation of $A$ may change the original eigenvalues and eigenvectors.

The mechanisms will be displayed in some very simple examples:

**Example 2**

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

The eigenvalues and corresponding eigenvectors of $A$ are:

$$\lambda_1 = 1, \quad \lambda_2 = 3$$

$$v_1 = (1, -1)^T, \quad v_2 = (1, 1)^T.$$  

Let

$$\tilde{A} = \begin{pmatrix} 2 & 1 + \varepsilon \\ 1 & 2 \end{pmatrix}, \quad \text{where } \varepsilon \text{ is some small number.}$$

$\tilde{A}$ then has the eigenvalues and eigenvectors

$$\tilde{\lambda}_1 = 2 - \sqrt{1 + \varepsilon} = 1 - \frac{\varepsilon}{2} + \frac{\varepsilon^2}{8} + \ldots$$

$$\tilde{\lambda}_2 = 2 + \sqrt{1 + \varepsilon} = 3 + \frac{\varepsilon}{2} - \frac{\varepsilon^2}{8} + \ldots$$

$$\tilde{v}_1 = (1, -\sqrt{1 + \varepsilon})^T = (1, -1)^T + (0, -\frac{\varepsilon}{2})^T + \ldots$$

$$\tilde{v}_2 = (1, \sqrt{1 + \varepsilon})^T = (1, 1)^T + (0, \frac{\varepsilon}{2})^T + \ldots.$$  

**Example 3**

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & \cdot \cdot \cdot \\ \cdot \cdot \cdot & \cdot \cdot \cdot & \cdot \cdot \cdot \\ 0 & 1 & 1 \end{pmatrix}$$

$$A + E = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & \cdot \cdot \cdot \\ \cdot \cdot \cdot & \cdot \cdot \cdot & \cdot \cdot \cdot \\ \varepsilon & 1 & 1 \end{pmatrix}$$

(The $(n, 1)$-element of $E$ is $\varepsilon$, and all other elements are zero.)

$A$ has all eigenvalues $\lambda_1 = \lambda_2 = \ldots = \lambda_n = 1$;

$A + E$ has eigenvalues $\lambda_\nu = 1 + \varepsilon (-1)^n \cos(2\nu\pi/n), \nu = 1, 2, \ldots, n.$
Here we have a perturbation of the eigenvalues of order $\varepsilon^{1/n}$. For example, $n = 10$ and $\varepsilon = 10^{-10}$ gives $|\lambda_{10}(A) - \lambda_{10}(A + \varepsilon)| = 0.1$.

**Example 4**

$$A = \begin{bmatrix} 20 & 0 & \cdots & 0 \\ 19 & 20 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 18 & \cdots & 20 \end{bmatrix} \quad A + \varepsilon = \begin{bmatrix} 20 & 0 & \cdots & 0 \\ 19 & 20 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \varepsilon & 18 & \cdots & 20 \end{bmatrix}$$

The characteristic equations are:

$$(20 - \lambda)(19 - \lambda) \cdots (1 - \lambda) = 0 \quad \text{and} \quad (20 - \lambda)(19 - \lambda) \cdots (1 - \lambda) = 20^{1/9} \varepsilon.$$

Thus $\lambda_{10}(A) = 10$ and $\lambda_{11}(A) = 11$.

However, $\varepsilon = 10^{-10}$ gives $\lambda_{10,11}(A + \varepsilon) \approx 10.5000 \pm 1.7334$. This example is taken from Ref. 1.

* * *

**General theorem on perturbations of eigenvalues:**

Let $\lambda$ be an eigenvalue of $A$ with multiplicity $p$. Consider perturbations of $A$ into $A + \varepsilon B$, where $B$ is a fixed matrix and $\varepsilon$ a sufficiently small parameter.

The $p$ eigenvalues of $A$ equal to $\lambda$ then fall into $m$ groups, where $m$ can be any integer between 1 and $p$ (depending on the choice of $B$). Let $r_1, r_2, \ldots, r_m$ denote the number of eigenvalues in each group (hence $\sum_{i=1}^{m} r_i = p$). The $r_\nu$ eigenvalues in a group, $\nu = 1, 2, \ldots, n$, then get perturbed to eigenvalues $\lambda_{\nu,k}(A + \varepsilon B)$, $k = 1, \ldots, r_\nu$, of $A + \varepsilon B$, with the following expansions:

$$
\lambda_{\nu,k}(A + \varepsilon B) = \lambda + C_{1,\nu} \varepsilon^{1/r_\nu} e^{i(2\pi/r_\nu)k} + C_{2,\nu} \left[ \varepsilon^{1/r_\nu} e^{i(2\pi/r_\nu)k} \right]^2 + \\
+ C_{3,\nu} \left[ \varepsilon^{1/r_\nu} e^{i(2\pi/r_\nu)k} \right]^3 + \cdots,
$$

where $C_{1,\nu}, C_{2,\nu}, \ldots$ are constants.

Thus, in the worst case when $m = 1$ and $r_1 = p$, the perturbations of the eigenvalue $\lambda$ are of order $\varepsilon^{1/p}$.

This theorem follows directly from a theorem for the roots of polynomials; see, for instance, Ref. 1.

* * *
Comments on the examples:

In Example 2 the errors in both eigenvalues and eigenvectors are power series in $\varepsilon$.

In Example 3, $r = p = n$ and we get a perturbation of the eigenvalues of the order $\varepsilon^{1/n}$. Note that in Example 3 the original matrix has only got one eigenvector. It can be shown that the perturbed problem has $n$ different eigenvectors which all differ from the original vector by the order $\varepsilon^{1/n}$.

In Example 4, the coefficients in the power series expansions for the eigenvalues of $A + \varepsilon E$ are very big, and for $\varepsilon = 10^{-12}$ these expansions are no longer valid!

**Theorem:**

Suppose $A$ has $n$ linearly independent eigenvectors. Let $\lambda_\nu(A)$, $\nu = 1, \ldots, n$, denote the eigenvalues of $A$, and $\lambda_1(A + \varepsilon B)$, $i = 1, \ldots, n$, the eigenvalues of a matrix $A + \varepsilon B$.

Then, for each $\lambda_i(A + \varepsilon B)$, $i = 1, 2, \ldots, n$,

$$\min_\nu |\lambda_\nu(A) - \lambda_1(A + \varepsilon B)| \leq \varepsilon \|B\| \|T\| \|T^{-1}\|,$$

where $T$ is a matrix with the eigenvectors of $A$ as columns. There are infinitely many possibilities of choosing $T$, since eigenvectors can be multiplied with non-zero constants.

This inequality holds for all the norms mentioned in Chapter I, Section 9, e.g. for $\|\cdot\|_2$. It motivates the definition of a condition number also for the eigenvalue problem.

**Definition:** The condition number $K(A)$ of $A$ with respect to eigenvalues is

$$\min \|T\| \|T^{-1}\|,$$

where $T$ contains the eigenvectors of $A$ as columns.

Since $\|T\| \|T^{-1}\| \geq \|TT^{-1}\| = 1$, we have $K(A) \geq 1$.

If the eigenvectors are orthogonal, they can also be chosen normalized and $T$ will then be a unitary matrix, and this will give $K(A) = 1$ for the norm $\|\cdot\|_2$. One can show that if $AA^* = A^*A$ then the eigenvectors are orthogonal, and $K(A) = 1$. If $A$ is Hermitian, i.e. $A = A^*$, then this condition is fulfilled and $K(A) = 1$.

Hence we come to the conclusion that: the eigenvalues of a Hermitian matrix are always well conditioned. An important special case is that of a real symmetric matrix.

**Example 5**

Both $A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1.001 \end{pmatrix}$ and $A_2 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ have an eigenvalue $\lambda_1 = 1$ with eigenvector $v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. 
Let 

\[ B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad \|B\|_2 = 1. \]

\[ A_1 + \varepsilon B = \begin{pmatrix} 1 & 0 \\ \varepsilon & 1.001 \end{pmatrix} \quad \text{gives } \lambda_1 = 1 \quad \text{and} \quad v_1 = \begin{pmatrix} 1 \\ -10^3 \cdot \varepsilon \end{pmatrix}, \]

\[ A_2 + \varepsilon B = \begin{pmatrix} 1 & 0 \\ \varepsilon & 2 \end{pmatrix} \quad \text{gives } \lambda_1 = 1 \quad \text{and} \quad v_1 = \begin{pmatrix} 1 \\ -\varepsilon \end{pmatrix}. \]

Here, both \( A_1 \) and \( A_2 \) are symmetric, which implies that the eigenvalues are well conditioned. When the perturbation \( \varepsilon B \) is added, the eigenvalues will move at most \( \varepsilon \|B\|_2 = \varepsilon \). But for \( A_1 \) the perturbation of an eigenvector is of the order of 1000 \( \varepsilon \). This happens because there is, for \( A_1 \), another eigenvalue close to \( \lambda_1 = 1 \).

***

Our conclusion from Example 5 is that even if the eigenvalues are well conditioned, the eigenvectors may be ill conditioned.

It can be shown that there is a general rule that

- close eigenvalues \( \Rightarrow \) ill-conditioned eigenvectors, and
- big condition number \( \Rightarrow \) ill-conditioned eigenvectors.

If a Hermitian (e.g. real symmetric) matrix has well-separated eigenvalues (relative to the norm of the matrix), then both eigenvectors and eigenvalues are well conditioned.

Note that, if the perturbation \( \varepsilon B \) is proportional to the magnitude of the numbers in the unperturbed matrix \( A \), e.g. \( \|\varepsilon B\| \leq \mu \cdot \|A\| \), the estimate for the perturbation of the eigenvalues given previously will be

\[ \min_{\lambda} \left| \lambda (A) - \lambda_1 (A + \varepsilon B) \right| \leq K(A) \cdot \mu \cdot \|A\|. \]

We cannot give any estimate of the relative error in any individual eigenvalue, but only an estimate of the error relative to the norm of the matrix \( A \).

3. **The QR Method**

Before the QR method for eigenvalue computations is introduced, we need some new concepts.

3.1 **Similarity transformations; Hessenberg matrices**

Let \( S \) be a non-singular \( n \times n \) matrix. If \( Ax = \lambda x \), then \( SAx = \lambda Sx \), and since \( S^{-1} \) exists, \( SAS^{-1} Sx = \lambda Sx \). Let \( B = SAS^{-1} \) and \( y = Sx \). We then get \( By = \lambda y \). Thus \( B = SAS^{-1} \) has the same eigenvalues as \( A \), but the eigenvector \( x \) of \( A \) is transformed into the eigenvector \( Sx \) of \( B \).
Definition: A transformation SAS\(^{-1}\) is called a similarity transformation. Similarity transformations preserve the eigenvalues.

Definition: An \(n \times n\) matrix \(A\) with \(a_{ij} = 0\) for \(i > j + 1\) is an upper Hessenberg matrix. It is of the form

\[
A = \begin{bmatrix}
x & x & x & \cdots & x \\
x & x & x & \cdots & x \\
x & x & \cdots & \cdots & x \\
\ddots & \ddots & \ddots & \ddots & \vdots \\
0 & x & \cdots & \cdots & x
\end{bmatrix}
\]

A lower Hessenberg matrix is defined analogously.

Theorem: Any \(n \times n\) matrix \(A\) can be transformed into an upper Hessenberg matrix with a similarity transformation \(B = QAQ^{-1}\). The matrix \(Q\) can be chosen as a unitary matrix.

Proof:

Step 1

Choose the \((n-1) \times (n-1)\) matrix \(\tilde{H}_i\) such that

\[
\tilde{H}_i = \begin{bmatrix}
a_{21} & 0 \\
a_{31} & 0 \\
\vdots & \vdots \\
a_{n1} & 0
\end{bmatrix}
\]

Here \(\tilde{H}_i\) can be taken as a Householder matrix, as is shown in Chapter II, Section 1.

Put \(H_i = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{H}_i \end{bmatrix}\); then \(H_i^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{H}_i^{-1} \end{bmatrix}\).
Here $x$ is used to denote elements which can be non-zero.

**Step 2**

Choose $\tilde{H}_2$ such that

$$
\begin{bmatrix}
  a_{12}^{(2)} \\
  \vdots \\
  a_{n2}^{(2)}
\end{bmatrix}
= \begin{bmatrix} x \\ 0 \\ \vdots \\ 0 \end{bmatrix}
$$

Put $H_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \tilde{H}_2 \end{bmatrix}$

Continue analogously.

We finally get the upper Hessenberg matrix $B = H_{n-2} \ldots H_2H_1H_1^{-1}H_2^{-1} \ldots H_{n-2}^{-1} = QAQ^{-1}$, where $Q = H_{n-2} \ldots H_2H_1$.

When Householder transformations are used, $Q$ is unitary (i.e. $Q^{-1} = Q^*$), since $H_1, H_2, \ldots$, are unitary.

(The required zeros at each step can also be produced by matrices of the same type as those used in Gaussian elimination. This, however, is not a unitary transformation.)

Q.E.D.

Suppose $A = A^*$. Let $B$ be the upper Hessenberg matrix $B = QAQ^*$ where $Q$ is unitary. $B^*$ will then be a lower Hessenberg matrix. But $B^* = (QAQ^*)^* = QA^*Q^* = QAQ^* = B$, which implies
that B must be tridiagonal. Hence the process described above transforms a Hermitian (e.g. real symmetric) matrix into a tridiagonal Hermitian matrix with the same eigenvalues as the original matrix.

The number of multiplications can give a rough measure of how "expensive" the transformations are:

General matrix $\rightarrow$ upper Hessenberg $\sim \frac{2}{3} n^3$ multiplications
Real symmetric matrix $\rightarrow$ tridiagonal $\sim \frac{1}{3} n^3$ multiplications

3.2 Brief description of the QR algorithm

The QR algorithm is an efficient method of finding all eigenvalues of a matrix. The method was developed in the early 1960's and superceded most other methods existing at the time.

The basic algorithm is the following:
Decompose $A_1 = A = Q_1 R_1$, where $Q_1$ is unitary and $R_1$ upper triangular.
Form the next matrix $A_2 = R_1 Q_1 = Q_1^T A Q_1$.
Decompose $A_2 = Q_2 R_2$.
Form the next matrix $A_3 = R_2 Q_2 = Q_2^T A_2 Q_2$, etc.

We get a sequence of matrices with the same eigenvalues as A (since we perform similarity transformations). Under certain conditions, this sequence converges to an upper triangular matrix with the eigenvalues as diagonal elements (see, however, the end of this Section 3.2).

One can show that
i) if A is upper Hessenberg, then $A_2$, $A_3$, ..., are upper Hessenberg;
ii) if A is real symmetric tridiagonal, then $A_2$, $A_3$, ..., are real symmetric tridiagonal.

Since Hessenberg (or tridiagonal) form is preserved by the QR method, and since this cuts down the amount of calculations considerably, compared to working on a full matrix, the matrix should first be transformed to this form before the QR method is applied. This is usually included in QR computer programs.

The eigenvalues tend to appear as

$$
\begin{bmatrix}
\lambda_1 & x & x & \cdots & x \\
& \lambda_2 & x & \cdots & x \\
& & \ddots & \ddots & \ddots \\
& & & \ddots & \lambda_{n-1} \\
& & & & \lambda_n \\
\end{bmatrix},
$$

where $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. The rate of convergence depends on the factors $(\lambda_{i+1}/\lambda_i)^{s}$, where $s$ is the number of iteration steps. To improve the convergence, shifts are always introduced.

The QR method with shifts:

Decompose $A + kV = QVR$ (I is the identity matrix and $k_V$ a constant. $k_V$ is subtracted from all eigenvalues.)
Form next matrix \( A_{n+1} = R \tilde{Q} A_n + k_n I \) \((k_n\) is added again to all eigenvalues.\)

Now, with shifts, the convergence depends on factors of the type

\[
\prod_{i=1}^{s} \frac{\lambda_{i+1} - k_n}{\lambda_{i} - k_n},
\]

where \( s \) is the number of iteration steps.

Usually an attempt is made to try to choose \( k_n = \lambda_n \) in order to get the smallest eigenvalue first. As mentioned above, \( a_n^{(s)} \) is likely to converge to \( \lambda_n \). Therefore, one of the following strategies for the choice of \( k_n \) can be adopted:

i) Take \( k_n = a_n^{(v)} \) in the current matrix \( A_n \).

ii) Take \( k_n = \) the smallest eigenvalue of

\[
\begin{bmatrix}
    a_{n-1,n-1}^{(v)} & a_{n-1,n}^{(v)} \\
    a_{n,n-1}^{(v)} & a_{nn}^{(v)}
\end{bmatrix}
\]

in \( A_n \).

When \( a_{n,n-1}^{(v)} \) is small enough, \( a_{nn}^{(v)} \) is accepted as an eigenvalue and the last row and the last column can be left out of the subsequent calculations (and the shifts are chosen such as to approximate \( \lambda_{n-1} \)).

If \( A \) is real but not symmetric, then it may have some complex eigenvalues. But the matrices \( A_2, A_3, \ldots \), generated by the QR method are all real (as long as the shifts \( k_n \) are real), and consequently the complex eigenvalues cannot appear in the diagonal. In this case, we get convergence to a matrix of the form

\[
\begin{bmatrix}
    x & x & x & \cdots & x \\
    x & x & \cdots & x \\
    & & & \ddots & \\
    x & \cdots & x & & x \\
    x & \cdots & x & & x \\
    & & \cdots & x & x \\
    0 & \cdots & x & & x \\
\end{bmatrix}
\]

where the complex eigenvalues can be computed from the \( 2 \times 2 \) "boxes" which appear at the diagonal. If a complex shift were introduced at any step, the resulting matrix would be complex. Complex arithmetic, however, requires much more computing time than real arithmetic,
and is therefore avoided whenever possible. Instead of a single complex shift \( k \), two steps 
with complex shifts \( k \) and \( \bar{k} \) can be made (the complex eigenvalues of a real matrix are com-
plex conjugate pairs), and this can be carried out in real arithmetic. (For a detailed de-
scription, see for example Ref. 2.).

Normally it takes about 2\( n \) QR steps to get all the eigenvalues. When an eigenvalue is 
found, the matrix is reduced (deflated), so that the transformations are performed on smaller 
and smaller submatrices.

The number of multiplications gives a good idea of the work involved in the different 
steps:

- General \( n \times n \) matrix \( \rightarrow \) upper Hessenberg \( \sim \frac{5}{4} n^3 \)
- One QR step on an \( n \times n \) upper Hessenberg matrix \( \sim 4 n^2 \)
- Real symmetric \( \rightarrow \) tridiagonal \( \sim \frac{2}{3} n^3 \)
- One QR step on an \( n \times n \) real tridiagonal matrix \( \sim c \times n \) where \( c \) is a constant which 
depends on the implementation of the method (e.g. one can get \( c = 13 \)).

Note

If we want only a few eigenvalues (< 40%) or if we have a large sparse matrix, other 
methods may be preferable.

4. COMPUTATION OF EIGENVECTORS

The most important method for eigenvector calculations is the inverse iteration, also 
called the Wielandt iteration. Since this method is a variant of the classical power method, 
this will be described first.

4.1 The power method

This method, in its classical form, gives the eigenvector corresponding to the eigen-
value with the largest absolute value. Suppose the \( n \times n \) matrix \( A \) has \( n \) linearly independent 
eigenvectors \( v_1, v_2, \ldots, v_n \) with corresponding eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_n \). Take a starting 
vector \( x_0 \). This vector can then, theoretically, be written as a linear combination of the 
vectors \( v_1, v_2, \ldots, v_n \) (but this expansion is of course not known).

Suppose \( x_0 = a_1 v_1 + a_2 v_2 + \ldots + a_n v_n \).

Compute \( x_1 = Ax_0 \).

Then \( x_1 = a_1 \lambda_1 v_1 + a_2 \lambda_2 v_2 + \ldots + a_n \lambda_n v_n \).

Compute \( x_2 = Ax_1 = a_1 \lambda_1^2 v_1 + a_2 \lambda_2^2 v_2 + \ldots + a_n \lambda_n^2 v_n \).

\( \vdots \)

\( x_k = Ax_{k-1} = a_1 \lambda_1^k v_1 + a_2 \lambda_2^k v_2 + \ldots + a_n \lambda_n^k v_n \).

If \( |\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n| \), then the term \( a_1 \lambda_1^k v_1 \) will eventually dominate in \( x_k \), 
and the sequence will converge to the eigenvector \( v_1 \) multiplied by some constant. The vectors 
\( x_2, x_3, \ldots \) can be divided by suitable factors, to ensure, for instance, that the largest 
component is always 1. When the process has converged, we shall have \( Ax_k = \lambda_1 x_k \). Hence we 
can also get an approximation to the largest eigenvalue \( \lambda_1 \), as well as to the eigenvector.

Note that the rate of convergence depends on \( |\lambda_2/\lambda_1| \). If \( |\lambda_1| = |\lambda_2| \) then the convergence
will be very slow, and if $|\lambda_1| = |\lambda_2|$ we might not get any convergence at all. Also, with an unfortunate choice of $x_1$, $\alpha_1$ may be small, and this also causes slow convergence.

Because of the bad convergence properties, the power method in its classical form is not to be recommended. But it is the theoretical basis for other, more useful, algorithms. The most important of these is inverse iteration, which will be described below. Some other methods achieve better convergence than the classical power method since they iterate with a set of $p$, $1 < p \ll n$, vectors, instead of just one.

4.2 Inverse iteration for eigenvectors

(Wielandt iteration)

Suppose we have computed an approximate eigenvalue $\tilde{\lambda}_1$ of $A$. We can assume that $\tilde{\lambda}_1$ is near to the exact eigenvalue $\lambda_1$, which has the corresponding eigenvector $v_1$. Then the matrix $A - \tilde{\lambda}_1 I$, where $I$ is the identity matrix, has an eigenvalue $\lambda_1 - \tilde{\lambda}_1$ with $v_1$ as eigenvector. Also, $v_1$ is an eigenvector of $(A - \tilde{\lambda}_1 I)^{-1}$ corresponding to the eigenvalue $1/(\lambda_1 - \tilde{\lambda}_1)$. Since $\tilde{\lambda}_1$ is close to $\lambda_1$ this eigenvalue will be very big, and therefore the power method applied on $(A - \tilde{\lambda}_1 I)^{-1}$ can be expected to converge rapidly. This turns out to work very well in practice, and one iteration is almost always enough to get the eigenvector correct to working precision, when the eigenvalue is so.

The power method implemented for $(A - \tilde{\lambda}_1 I)^{-1}$ is called inverse iteration, or sometimes Wielandt iteration. The matrix $(A - \tilde{\lambda}_1 I)^{-1}$ is not computed, but the successive linear systems

$$(A - \tilde{\lambda}_1 I)x_{\mu+1} = x_\mu, \quad \mu = 0, 1, \ldots,$$

where $x_0$ is the chosen starting vector, are solved by PLU decomposition of $A - \tilde{\lambda}_1 I$. Special care has to be taken when these systems are solved, since the matrix is almost singular (and over- or underflow may occur in the computer arithmetic).

5. THE COMPUTATION OF THE SINGULAR VALUE DECOMPOSITION

The singular value decomposition was introduced in Chapter II, Section 7.2. A brief repetition will be given here, but see also the above-mentioned section for notations and explanations.

An $m \times n$ matrix $A$, $m \geq n$, can be decomposed as

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^*,$$

where $U$ is an $m \times n$ unitary matrix, $V$ is an $n \times n$ unitary matrix, and $\Sigma$ is an $n \times n$ diagonal matrix

$$\Sigma = \begin{bmatrix} 
\sigma_1 & & \\
& \sigma_2 & \\
& & \ddots \\
& & & \sigma_n
\end{bmatrix}$$
[This decomposition can also be done for $m < n$, in which case the middle matrix is of the form $(\Sigma \ 0)$.]

The numbers $\sigma_1, \sigma_2, \ldots, \sigma_n$ are the singular values of $A$. They are all greater than or equal to 0; $\sigma_i^2$, $i = 1, 2, \ldots, n$ are the eigenvalues of $A^*A$. It can be shown that $V$ must have the eigenvectors of $A^*A$ as columns and $U$ must have the eigenvectors of $AA^*$ as columns.

If $A = U(\Sigma) V^*$ then, for an $m \times m$ unitary matrix $Q_1$ and an $n \times n$ unitary matrix $Q_2$,

$$B = Q_1AQ_2^* = Q_1U(\Sigma) V^*Q_2^* = U_1(\Sigma)V_1^*,$$

where $U_1 = Q_1U$ and $V_1 = Q_2V$ are unitary; $U_1(\Sigma)V_1^*$ is the singular value decomposition of $B$, and the singular values are the same as those of $A$. Hence, pre- and post-multiplication by unitary matrices does not change the singular values. This is used in a computational algorithm, where the matrix $A$ first is transformed into a matrix $B = Q_1AQ_2^*$, where $B$ has the form

$$B = \begin{bmatrix}
  x & x & 0 \\
  x & x & \\
  & \ddots & \ddots \\
  & x & x \\
  0 & & x
\end{bmatrix}$$

The transformation of $A$ into $B$ is done with Householder transformations. First, the first column is transformed by pre-multiplication by a suitable matrix, then the new first row is transformed by post-multiplication, then the second column, second row, etc. Then the eigenvalues of $B^*B$ must be computed. This is done with the QR method, but implemented in such a way that the matrix $B^*B$ is not computed. The method works directly on the matrix $B$. If not only the singular values but also the matrices $U$ and $V$ are required, the latter can be obtained by accumulation of the transformations.

A detailed description of the method and an Algol program has been given by Golub and Reinsch④.
REFERENCES


Some literature on iterative methods and sparse matrices (not treated in this report):
<table>
<thead>
<tr>
<th>Subject Index</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back substitution</td>
<td>I.3</td>
</tr>
<tr>
<td>Balancing (scaling, equilibration)</td>
<td>I.5</td>
</tr>
<tr>
<td>Band matrices</td>
<td>I.11</td>
</tr>
<tr>
<td>Characteristic polynomial</td>
<td>III.1</td>
</tr>
<tr>
<td>Cholesky decomposition</td>
<td>I.7</td>
</tr>
<tr>
<td>Condition number</td>
<td></td>
</tr>
<tr>
<td>for eigenvalues and eigenvectors</td>
<td>III.2</td>
</tr>
<tr>
<td>for inversion</td>
<td>1.9</td>
</tr>
<tr>
<td>for the least squares problem</td>
<td>II.6</td>
</tr>
<tr>
<td>Crout's method</td>
<td>I.6</td>
</tr>
<tr>
<td>Doolittle's method</td>
<td>I.6</td>
</tr>
<tr>
<td>Eigenvalues</td>
<td>III.1</td>
</tr>
<tr>
<td>Eigenvectors</td>
<td>III.1</td>
</tr>
<tr>
<td>Equilibration (scaling, balancing)</td>
<td>I.5</td>
</tr>
<tr>
<td>Error matrix</td>
<td>II.4</td>
</tr>
<tr>
<td>Gaussian elimination</td>
<td>I.3</td>
</tr>
<tr>
<td>Gram-Schmidt orthogonalization, classical and modified</td>
<td>II.7.1</td>
</tr>
<tr>
<td>Hermitian matrix</td>
<td>I.7</td>
</tr>
<tr>
<td>Hessenberg matrix</td>
<td>III.3.1</td>
</tr>
<tr>
<td>Householder transformations</td>
<td>II.1</td>
</tr>
<tr>
<td>Ill-conditioning</td>
<td>I.8</td>
</tr>
<tr>
<td>Inverse iteration (Wielandt iteration)</td>
<td>III.4.2</td>
</tr>
<tr>
<td>Iterative improvement</td>
<td>I.10.2</td>
</tr>
<tr>
<td>Iterative methods for solution of linear systems</td>
<td>I.11</td>
</tr>
<tr>
<td>Least squares problems, linear</td>
<td>II.2</td>
</tr>
<tr>
<td>Linear systems of equations</td>
<td>I.1</td>
</tr>
<tr>
<td>Lower triangular matrix</td>
<td>I.3</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>I.3</td>
</tr>
<tr>
<td>Moore-Penrose generalized inverse</td>
<td>II.7.2</td>
</tr>
<tr>
<td>Multiplicity of eigenvalues</td>
<td>III.1</td>
</tr>
<tr>
<td>Normal equations</td>
<td>II.2</td>
</tr>
<tr>
<td>Norms</td>
<td>I.9</td>
</tr>
<tr>
<td>Orthogonal matrix</td>
<td>II.1</td>
</tr>
<tr>
<td>Topic</td>
<td>Section</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Permutation matrix</td>
<td>I.4.1</td>
</tr>
<tr>
<td>Pivoting</td>
<td>I.4</td>
</tr>
<tr>
<td>PLU decomposition</td>
<td>I.4.1</td>
</tr>
<tr>
<td>Positive definite matrix</td>
<td>I.7</td>
</tr>
<tr>
<td>Power method</td>
<td>III.4.1</td>
</tr>
<tr>
<td>QR algorithm</td>
<td>III.3.2</td>
</tr>
<tr>
<td>QR (QR) decomposition</td>
<td>II.3</td>
</tr>
<tr>
<td>Rank of a matrix</td>
<td>II.2</td>
</tr>
<tr>
<td>Scaling (balancing, equilibration)</td>
<td>I.5</td>
</tr>
<tr>
<td>Similarity transformation</td>
<td>III.3.1</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>II.7.2  and III.5</td>
</tr>
<tr>
<td>Singular values</td>
<td>II.7.2</td>
</tr>
<tr>
<td>Sparse matrix</td>
<td>I.11</td>
</tr>
<tr>
<td>Subordinate matrix norm</td>
<td>I.9</td>
</tr>
<tr>
<td>Symmetric matrix</td>
<td>I.7</td>
</tr>
<tr>
<td>Trace of a matrix</td>
<td>III.1</td>
</tr>
<tr>
<td>Transpose of a matrix</td>
<td>I.7</td>
</tr>
<tr>
<td>Triangular matrix</td>
<td>I.3</td>
</tr>
<tr>
<td>Unitary matrix</td>
<td>II.1</td>
</tr>
<tr>
<td>Upper triangular matrix</td>
<td>I.3</td>
</tr>
<tr>
<td>Wielandt iteration (inverse iteration)</td>
<td>III.4.2</td>
</tr>
</tbody>
</table>