CS46

TODAY'S COMPUTATIONAL METHODS OF LINEAR ALGEBRA

BY

GEORGE E. FORSYTHE

TECHNICAL REPORT NO. CS46
AUGUST 11, 1966

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

TODAY'S COMPUTATIONAL METHODS OF LINEAR ALGEBRA[†]

by

George E. Forsythe[††]

CONTENTS

[†] Invited address presented 13 May 1966 to a national meeting of SIAM at Iowa City, sponsored by the Air Force Office of Scientific Research. The preparation of the manuscript was sponsored by the Office of Naval Research under Contract Nonr-225(37) (NR-044-211) Received by the editors

[††] Computer Science Department, Stanford University, Stanford, Calif. 94305.

1. Introduction. This survey of selected computational aspects of linear algebra is addressed to members of SIAM who are not specialists in numerical analysis. The reader is assumed to have a general familiarity with the algebra and analysis of finite vectors and matrices, including norms, and to know the Gaussian elimination process. A completely adequate background is given in the first 72 pages of Faddeeva [9]. A much more complete background for practical matrix work is found in Bellman [3], Marcus and Minc [38], and Wilkinson [61].

Far more extensive expositions of the computational methods of linear algebra are to be found in Fox [14], Noble [42], Householder [28], and Wilkinson [61].

The author gratefully acknowledges conversations with Gene H. Golub, Richard Hamming, and William Kahan, and especially the opportunity to see a draft of Kahan [32]. He also acknowledges substantial debts to Cleve Moler for the use of material from Forsythe and Moler [12].

2. Computational problems of linear algebra. The ordinary computational problems of linear algebra are concerned with matrices of real numbers.

a. Let A be an n-rowed, n-columned matrix of real numbers. Let b be an n-rowed column vector of real numbers. The traditional linear-equations problem is to find an n-rowed column vector x such that

$$A x = b .$$

It is normally assumed that A is a nonsingular matrix, since then and only then does a unique solution exist for all b.

b. With the same A as in part a, another traditional problem is to find the inverse matrix A^{-1}.

c. Let A be an n-rowed, n-columned matrix of real numbers which is symmetric. The third traditional problem is to find some or all of the (necessarily real) eigenvalues of A. Recall that an eigenvalue of A is a number λ for which

there exists a column vector u such that

$$A u = \lambda y .$$

Such a vector u is called a (column) eigenvector of A belonging to λ, and often the computational problem includes finding a u belonging to each eigenvalue computed. There exist n orthonormal eigenvectors of A, one belonging to each eigenvalue of A.

d. Let A be an unsymmetric n-rowed, n-columned matrix of real numbers. Another traditional problem of linear algebra is to find some or all of its eigenvalues, and sometimes also its corresponding column eigenvectors and row eigenvectors. Recall that a row eigenvector belonging to λ is an n-columned row vector v such that

$$v A = \lambda v .$$

When A is not symmetric, the problem is complicated in many ways: First, some of the eigenvalues λ are ordinarily complex numbers. Second, there may not exist n linearly independent column eigenvectors, and those which exist are not usually orthogonal. Indeed, they are likely to be nearly linearly dependent and the same holds for the row eigenvectors. Third, if an eigenvalue λ is a root of multiplicity $k > 1$ of the characteristic equation $\det(A - \lambda I) = 0$, then there may exist anywhere from 1 to k linearly independent column eigenvectors belonging to λ. (If A were symmetric, there would always be k.) If the number is less than k, it corresponds to one or more nondiagonal blocks in the Jordan canonical form of A, or equivalently to so-called nonlinear elementary divisors of A. Fourth, multiple or nearly multiple eigenvalues of A are likely to be very rapidly changing functions of the elements $a_{i,j}$ of A, so that computations are at best tricky.

e. For any column vector y, define the p-th power norm of y to be

$$\|y\|_p = \left(\sum_{i=1}^{n} |y_i|^p \right)^{\frac{1}{p}} \quad . \tag{1}$$

Here p is a real number with $1 \leq p < \infty$, and $y_1, \ldots, y_n$ are the components of y in a given coordinate system. We define the maximum norm as the limiting case $p \to \infty$ of (1);

$$\|y\|_\infty = \max_{1 \leq i \leq n} |y_i| \quad . \tag{2}$$

The norms most used in numerical analysis are $p = 1, 2, \infty$, but statisticians are now giving attention to values of p between 1 and 2.

Let A be an n-rowed, k-columned matrix of real numbers, and let b be an n-rowed column vector. Given some p, a more recent computational problem is to find a k-rowed column vector x such that

$$\| A x - b\|_p \quad \text{is minimized} \quad .$$

When $p = 2$, the usual case, this is the linear least-squares problem. For $p = 2$ the unit sphere in the norm is very smooth, and methods of analysis work well. However, for $p = 1$ or ∞ the unit sphere has many corners, and methods of minimizing $\|Ax - b\|_p$ become combinatorial or discrete.

f. For two n-rowed column vectors x and y, we define $x \geq y$ to mean that $x_i \geq y_i$ for all components of x and y.

Let A and b be as in part e above. Then an important computational problem is to describe the set S of k-rowed column vectors x such that

$$A x \geq b \quad .$$

Sometimes, as in linear programming problems, one looks for vectors x in S such that $c^T x$ is a minimum, where c is a given k-rowed column vector.

So far we have spoken only of matrices of real numbers. Similar problems are posed occasionally for matrices of complex numbers. Many of the problems can also be phrased for matrices whose elements are expressions in indeterminates or letters. As methods of symbol manipulation on digital computers become more accessible to computer users, problems of linear algebra with matrices of letters will be studied more. Practical symbol manipulation will probably do more to interest mathematicians in computing than anything that has happened in the computer era to date.

The present discussion is limited to matrices of numbers, and moreover to problems a, b, c, d. For discussions of problem e with $p = 2$, the reader is referred to Golub and Kahan [18]. For problem f see presentations on linear programming like Dantzig [5].

Why do the linear problems a, b, c, and d arise so often? Why are they important? The answer is that linear operators are the simplest ones in mathematics, and the only operators that are fully understood in principle. Hence they are a natural model for an applied mathematician to use in attacking a problem. Even though linear operators in infinite-dimensional spaces will occur in analysis of differential equations (for example), the realities of computing mean that only finite-dimensional spaces can be handled with digital computers.

More realistic models of applied mathematics are usually nonlinear. But, whenever nonlinear operators are used, the actual solution of functional equations almost always involves the approximation of nonlinear operators by linear ones. A typical example of this is the use of Newton's method for solving

a system of nonlinear equations, in which at every step a locally best-fitting linear equation system must be solved. Nonlinear problems usually are very hard In attacking them by linear methods, it is essential that our linear tools be very sharp, so that they can be relied upon to work without failure. Only in this way can the analyst concentrate on the real difficulties of the nonlinear world. This point of view not only emphasizes the importance of being able to solve linear problems, but also the necessity of solving linear systems with extremely reliable methods.

Linear equation systems a arise directly mainly from two sources. One is from an approximation to linear functional equations, usually ordinary or partial differential equations. The other source is a problem of data fitting, interpolation, or approximation by linear families of functions.

Eigenvalue problems usually arise from studies of vibration or stability or resonance of linear physical systems (e.g., flutter of aircraft and criticality of reactors), or from factor analysis problems.

An excellent textbook by Noble [42] gives a number of physical examples of computational matrix problems.

3. A closer look at the problems. Since actual computers have finite storage capacity and a finite precision, we need to have a closer look at the nature of the matrices A and the computational problems.

Is the matrix A dense (most elements $a_{i,j} \neq 0$), or is it sparse (most elements $a_{i,j} = 0$)? If A is sparse, do the nonzero elements form a significant pattern? For example, is A triangular ($a_{i,j} = 0$ for $i > j$ or for $i < j$)? Is it of Hessenberg form ($a_{i,j} = 0$ for $i > j + 1$ or for $j > i + 1$)? Is it a band matrix ($a_{i,j} = 0$ for $|i - j| > m$, where $m \ll n$? Is it a tridiagonal matrix (i.e., a band matrix with $m = 1$)? All these special forms occur frequently, and can be given special consideration.

Is the matrix A symmetric? Positive definite? If it is sparse, is the pattern associated with the adjacency matrix of some graph? Frequently matrices associated with structures or with partial difference equations are best understood in terms of the associated graph.

Are the elements $a_{i,j}$ stored in the computer memory, to be retrieved when needed, or are they regenerated from some algorithm, as needed? One might define the informational content of a matrix as the number of cells needed (on a certain computer) to store the data and program to obtain all the $a_{i,j}$. The author knows of no work on this concept, which is clearly relevant to matrix computation.

What is the size of the matrix A, relative to the memory size and speed of a given computer?

If we are solving a linear equation system $Ax = b$, do we have many different right-hand sizes b, or just one? Do we have many different matrices that are close together, or do we have just one A? Are the elements of A are precise mathematical numbers (for example, integers), or are they physical numbers subject to uncertainty? Any uncertainty in A and b leads to uncertainty in the definition of x as the solution of $Ax = b$. What x does the problem's proposer want to see? Even when A and b are mathematical numbers, the solution x is normally not representable as a finite-precision number in the computer's number base. Of the various approximate answers x which might be obtained, what is the proposer's desire? For example, does he want $\|x - A^{-1}b\|$ to be small, where $A^{-1}b$ is the true answer? Or would the proposer settle for an x such that $\|Ax - b\|$ is small? For each case: which norm, and how small?

Most proposers of linear equation systems haven't considered these questions, and look to the numerical analyst to explain the possibilities and select the options.

If a proposer requests the inverse matrix A^{-1}, it is usually worth finding out why. Frequently he merely wishes a convenient way to solve $Ax = c$ for an arbitrary vector c. Having A^{-1} stored away, the proposer expects to obtain the solution x in the form $A^{-1}c$, for any new c that comes along. It should be pointed out that there are other ways to obtain $A^{-1}c$ for new vectors c, ways that require no more storage and take no longer for the same accuracy, than the multiplication of A^{-1} by c. Because of these facts, the computation of A^{-1} may frequently be dispensed with. However, certain statistical applications really do require knowledge of at least the diagonal elements of A^{-1}.

The eigenvalue problem c for symmetric matrices A can require finding all the eigenvalues, or only a few. It matters a good deal whether or not the corresponding eigenvectors are needed. If a complete set of eigenvectors is needed, is it important that they be orthogonal to each other? Getting orthogonal eigenvectors corresponding to multiple eigenvalues is far more difficult than just getting eigenvalues.

In the eigenvalue problem d for nonsymmetric matrices A, one has similar choices: do we want all eigenvalues, or just some? Do we want column eigenvectors? Do we want row eigenvectors? Both? But then comes a new choice. If some eigenvalues are multiple and correspond to a nonlinear elementary divisor, what vectors does the proposer want to see? In monographs on algebra one learns about chains of principal vectors that with the eigenvector form a basis for the null space N of $(A - \lambda I)^k$, where λ is an eigenvalue of multiplicity k with an elementary divisor of degree k. These principal vectors are associated with the Jordan canonical form of A. It is my impression that a proposer who has a good background in algebra will want to see a set of principal vectors (they are not unique). But these principal vectors are extremely hard to compute,

partly because they are discontinuous functions of the data. It is likely that an orthogonal basis for the nullspace N would be a more useful set of vectors. The matter seems to be poorly understood by problem proposers and numerical analysts.

Matrices with actual multiple eigenvalues are very rare, and a small computational perturbation of these will normally destroy the equality of eigenvalues. One might therefore assume that we need not be concerned in practice with what to do about them. But, in fact, the bad behavior of nonlinear divisors carries over in practice to a surprisingly large set of neighboring matrices. These neighboring matrices have distinct eigenvalues, but the k column eigenvectors are so nearly linearly dependent that they cannot be separated in a normal computation. So also here one faces the problem of what vectors to give the proposer.

In a least squares problem, say a search for x to minimize $f(x) = \|Ax - b\|_2$, does the proposer really want a minimum of $f(x)$, or does he merely wish an x that gives a value of $f(x)$ fairly close to the minimum? In a curve-fitting problem, for example, one can often get a surprisingly good fit by a polynomial with coefficients very different from those of the minimizing polynomial.

In all of the above computational problems, it is important to ascertain which of the following types of answers the problem proposer is looking for:

a) a surmised answer, with no estimates of its correctness;

b) some answer, together with some sort of probabilistic assertions about its correctness;

c) some answer, together with mathematically provable bounds for its error.

Normally it is more expensive to obtain b) than a), and still more expensive to obtain c).

It is not obvious which of the above types of answer the problem proposer will want. Frequently a) is quite satisfactory. The physical scientist and engineer frequently have their own checks on the validity of an answer, and may neither need nor wish the mathematician's rigorous bounds. They may recognize, for example, that the mathematical model is such a rough approximation to reality that mathematical bounds would only be ludicrous. When mathematicians enter the practical world of engineering, the rules by which mathematics is played frequently have little relevance. Numerical analysts frequently have trouble deciding when to play the game according to mathematician's rules and when to play it like engineers. It is, of course, extremely pleasant to encounter those occasional examples where mathematically provable bounds can be found that are just as accurate and cheap as surmised answers. One should never cease looking for such miracles, because they do occur! One has been just reported at this SIAM Symposium; see Fox, Henrici, and Moler [26].

4. Nature of computer hardware and software. The character of achievable solutions to the computational problems of linear algebra is greatly influenced by the nature of the computing systems available to us. It is customary to separate computer systems along the following lines:

a) Computer hardware--the nature of the electronic circuitry of a computer;

b) Computer languages--the languages in which are described algorithms for the solution of a given problem on a given computer;

c) Computer software--the programs which make it possible for a computer actually to perform the algorithms described in the computer language.

In looking at computer hardware for computations in linear algebra one wants to know what precision is available for computation--how many digits are

in the significand of the floating-point operands, and to what base? One is also interested in the cost and speed of double-precision operations. In matrix algebra work the critical operation is frequently the computation of a rounded single-precision approximation to the double-precision inner product of two vectors whose components are single-precision floating-point numbers. The speed and cost of this inner product are very important.

One wonders whether the hardware rounds the result of an arithmetic operations, or whether it is chopped off. Best of all is a system that lets the programmer decide when to round and when to chop.

What happens when the result of an arithmetic operation exceeds the capacity of the floating-point system? Are there "traps" which make it possible for the system to detect overflow or underflow? Can these traps be by-passed, (turned off) by the programmer? When an overflow or underflow is detected, is all essential information recoverable, so that the solution can continue? Or are vital bits of information irretrievably lost?

What is the exact nature of the arithmetic operations in the machine? If one is to prove theorems about the behavior of a computation, one needs certain properties of the arithmetic. Because of the rounding of the machine, it is well known that addition and multiplication are not associative, nor are they distributive. Nevertheless, one can do surprisingly good analysis, provided only that the arithmetic is monotonic.

By multiplication being monotonic, we mean, for example, that if $0 < a < b$ and $0 < c$. then $a \times c \leq b \times c$. Such properties seem elemental, but they are extremely helpful. And they are surprisingly often absent!

It must be noted that apparently minor changes in the hardware of the arithmetic circuitry can make surprisingly large differences in the behavior of the algorithms.

A great many computer languages have been devised for the description of scientific algorithms. These range from the very elementary codes for Turing machines, through the machine codes of computers, to various algebraic languages like the forms of Fortran, Algol, and PL/I. All these languages are equivalent, in the sense that the class of representable algorithms is the same for all of them. The languages differ only in regard to human convenience and in the compilation problems they create. Can one conveniently represent such a data structure as a triangular matrix in a certain language? In typical languages like Algol or Fortran, one must choose between representing it as part of a much larger square matrix, on the one hand, or as an artificially created one-dimensional array, on the other. The former choice is humanly convenient and wastes space; the latter choice saves the computer time and space, at the cost of confusing the human.

Most matrix algorithms have "inner loops" where most of the computing time is spent. If only this inner loop is programmed very efficiently in machine code, the program will run very rapidly. It scarcely matters how the rest of the algorithm is programmed. Hence a very important question for any algebraic language is whether it is easy to incorporate pieces of machine code into them. Perhaps the question is more appropriately addressed to the software system that translates the algebraic language into machine code.

Another important property of a computer language is its readability by human beings. If the algorithm is correctly written, a computer will (practically always read it correctly. But the practical use of the algorithm depends on the ability of human beings to comprehend it, adapt it to other uses, improve it in the light of recent discoveries, and so on. The human readability of existing languages differs a great deal.

The most important software programs for the scientific computer user are the monitors and the compilers. The compilers are vast symbol-manipulation programs that translate an algorithm from, say, Fortran to the machine code of a given computer. Compilers should be distinguished from the languages they translate, and yet of course compilers and languages influence each other. Compilers differ greatly in speed, in the optimality of the machine code produced in the translation, and in the diagnostic facilities offered.

As we noted above, it is important that compilers be able to accept pieces of algorithms written in machine code, and incorporate them into a program otherwise written in an algebraic language. For matrix work, the ability to compile fast codes for iterative loops (the for statment of Algol) is very important.

Most compilers are now imbedded in control programs variously called master control programs, monitor systems, or operating systems. These monitor systems generally retain ultimate control of a computer, preventing a possibly erroneous user program from consuming vast amount of unwanted time, or from damaging the monitor system or other persons' programs by illegal assignments. Also, the monitor systems generally recover control of the machine in case of overflow or underflow. This is a point of much interest to writers of linear algebra programs. In case of overflow or underflow, what happens next? Can the linear algebra program recover control of the computer and repair the damage done by the overflow or underflow? (This assumes that the hardware retains the necessary information.) Or does the monitor system take over the machine and ruthlessly flush the offending program from the machine? If the latter occurs, then extra time must be taken in each program to make sure that overflow or underflow cannot occur.

5. The state of the art, 1953 and now. It is safe to say that matrix computation has passed well beyond the stage where an amateur is likely to think of computing methods which can compete with the better known methods. Certainly one cannot learn theoretical linear algebra and an algebraic programming language, and nothing else, and start writing programs which will perform acceptably by today's standards. There is simply too much hard-earned experience behind the better algorithms, and yet this experience is hardly mentioned in mathematical textbooks of linear algebra.

The amount of literature on matrix computations is staggering. In 620 pages, Faddeev and Faddeeva [8] record a pretty complete account of computational methods up to around 1958. In 662 pages, Wilkinson [61] gives most of what is known about computing eigenvalues of dense, stored matrices (both symmetric and unsymmetric), with error bounds for many algorithms. There is very little overlap between the two books, because Wilkinson and a few contemporaries created most of the material in his book in the years after 1958. No one could possibly start research in the numerical mathematics of linear algebra without a thorough knowledge of the relevant material in these books.

It is perhaps instructive to examine the state of matrix computation in 1953, when the author wrote a survey [10] of methods for solving linear systems at the Institute for Numerical Analysis of the National Bureau of Standards, Los Angeles. We were amateurs. For dense, stored matrices we knew Gaussian elimination, of course. We knew that it sometimes produced quite poor results. We weren't always sure why. We debated endlessly about how to pick pivots for the elimination, without settling it. The debate still continues, but now mainly among persons who don't understand that the main lines of the answer have been settled. Because of misunderstood difficulties with Gaussian elimination, we searched for other methods which might do better.

The conjugate-gradient method had been devised for sparse matrices by Lanczos [36], and Hestenes and Stiefel [27]. In [10] I guessed that it might also prevail for dense, stored matrices, despite the extra time it would require, because we understood how to use higher precision to make the conjugate-gradient method work well. We did not realize that the same higher precision and a proper pivotal strategy would make Gaussian elimination work. We were not quite aware of the extent of problems of ill conditioning of matrices.

The only analysis available to us was the monumental work of von Neumann and Goldstine [41, 20]. They avoided the pivoting problem by reducing any regular linear equation system $Ax = b$ to the positive definite system $A^TA\ x = A^Tb$. We knew that this normalization of the problem was costly in time and worsened the condition of the problem. Von Neumann and Goldstine presented guaranteed error bounds for the solution; actually observed errors were found to be perhaps 100 times smaller in reasonable cases. The form of the error analysis was a direct comparison of machine arithmetic with exact operations. The nonassociativity and nondistributivity of machine arithmetic made the analysis extremely difficult. In any case, it could only handle scaled fixed-point arithmetic. Because of the size of their error bounds, von Neumann and Goldstine were unnecessarily pessimistic about the possibility of inverting general matrices of orders over 15 on machines with the 27-bit precision of the IBM 7090 series.

For the eigenvalue problems, things were in much worse state. We had the power method with matrix deflation. While reasonably satisfactory for a few dominant roots, its general application requires intuition and luck, and defies a complete algorithmization. For dense, stored symmetric matrices we had the 1846 method of Jacobi [31], rediscovered and analyzed by Goldstine, Murray and von Neumann [19], and it was quite satisfactory. Givens was writing up

his newly discovered method, maybe 7 to 9 times faster than Jacobi's and a basic step toward currently used methods.

For nonsymmetric matrices, things were ghastly. If the power method wouldn't work, we had practically no alternatives. We could search for zeros of $\det(A - zI)$ in some manner or another. We bravely tried methods for determining the characteristic polynomial, as described in Faddeeva [9], and found them to be hopeless. It was almost unbelievable, how badly the standard methods for $n = 4$ would perform for $n = 10$. Lanczos was advocating his new method of finite iterations, which became the source of modern methods in a later line of development through the Stiefel and Rutishauser QD-algorithm, (see Rutishauser [50] and Henrici [25]), the LR-algorithm of Rutishauser [51], and the QR algorithm of Francis [15, 16] and Kublanovskaja [35]. However, the original Lanczos method needed careful management, because the raw results were often poor.

6. The linear equations problem. For large, sparse matrices, like those arising in finite-difference approximations to partial differential equations, there is a whole special literature. See Varga [57], Forsythe and Wasow [13], the work of David Young, Jim Douglas Jr., Stiefel, and many others. The methods seem to depend for their success on the nature of the continuous problem being approximated. Because the matrices are sparse, the prevailing methods are iterative. I shall omit further discussion of them, and confine attention to dense, stored matrices.

For a general matrix A, the solution of the linear system $Ax = b$ by Gaussian elimination requires $n^3/3 + \underline{O}(n^2)$ multiplications, and the same number of additions. Recently Klyuyev and Kokovkin-Shcherbak [34] proved that

no method using rational operations for general A, b can take fewer operations. This result had long been believed but not proved. The result has two consequences:

(i) Gaussian elimination is likely to remain the method of choice for solving dense linear systems, when it works, because it is as fast as any.

(ii) The solution of a linear system of large order n is going to require a very substantial amount of computing time, at least for serial computers. For $n = 1000$, we have $1/3 \times 10^9$ multiplications and additions. If we can multiply and add in 10 microseconds, we need 3333 seconds, or about an hour of computation. In fact, there is some overhead also, and on an IBM 7094 (Model II) the solution would take over 2 hours. However, the storage of the million elements of data requires extensive use of some bulk storage like tapes or disks, as only some 20,000 elements or so can be kept in the current 32,000-word core storage. The very numerous transfers of matrix elements from core to magnetic tapes appear likely to wear out the tapes before the solution can be obtained, according to certain tests made at Stanford! I know of no comparable experience with magnetic disks or other form of bulk storage.

As a result, we cannot consider order $n = 1000$ to represent a practical linear equations problem, but we will undoubtedly soon be able to do it regularly for perhaps $500.

The case $n = 100$ is now easy and costs around $1 on an IBM 7094. The case $n = 10,000$ is likely not to be accessible for a long time, and it would take over 2000 hours now on an IBM 7094.

There is beginning to be serious consideration of computers with a substantial amount of parallel operation, so that perhaps much of the solution of a linear system could be done simultaneously. Preliminary studies make it clear that the solution of a linear system could very easily make use of parallel

computation, if it should prove worth while. Apparently only $O(n)$ operation times would be needed for solving a linear system, if one had a sufficiently large amount of parallel arithmetic circuitry.

7. Inherent inaccuracy in solutions of linear systems. Given a nonsingular matrix A and a nonzero b, let x be the solution of $Ax = b$. Suppose A and b are subject to uncertainty. What is the resultant uncertainty in the solution x?

For any column vector y of order n, define $\|y\|$ to be the euclidean length of y:

$$\|y\| = \|y\|_2 = \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2} \quad .$$

For any n-by-n square matrix A, define the spectral norm $\|A\|$ by

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

These functions $\|...\|$ give useful measures of the size of vectors and matrices, respectively.

For a nonsingular matrix A, define the condition of A, $\mathrm{cond}(A)$ by the relation

$$\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\| \quad .$$

The concept of condition of a matrix seems to have been introduced by Turing [55], and studied extensively by Todd ([53] and some later papers) and many others.

One of the main uses of the concept of condition lies in answering the question posed at the start of this section. Suppose that A is known exactly, but that b is subject to uncertainty. Let $x + \delta x$ solve the system with matrix A and right-hand side $b + db$. Then

$$
\begin{aligned}
A(x + \delta x) &= b + db \ ; \\
x + \delta x &= A^{-1}b + A^{-1}db \ ; \\
\delta x &= A^{-1}db \ ; \\
\|\delta x\| &\leq \|A^{-1}\| \cdot \|db\| \ .
\end{aligned} \tag{3}
$$

Since $A\,x = b$, we have

$$\|b\| = \|Ax\| \leq \|A\| \cdot \|x\| \ . \tag{4}$$

Dividing (3) by (4), we have

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \cdot \|A^{-1}\| \ \frac{\|db\|}{\|b\|} \ ,$$

or

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \ \frac{\|db\|}{\|b\|} \ . \tag{5}$$

Inequality (5) shows that the relative uncertainty in x is bounded by $\text{cond}(A)$ times the relative uncertainty in b. The bound in (5) is attainable, for any nonsingular A and nonzero b. This is easy to see, if we perform a change of coordinates in which A takes a diagonal form.

As a linear transformation, A takes vectors x into vectors b. A fundamentally important, but too little known theorem states that by a certain orthogonal change of coordinates in the space of x, and by another orthogonal change of coordinates in the space of b, the matrix A can be put in the diagonal form

$$A = \begin{bmatrix} \mu_1 & & & 0 \\ & \mu_2 & & \\ & & \ddots & \\ 0 & & & \mu_n \end{bmatrix} .$$

Here the positive numbers $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$ are called the <u>singular values</u> of A. Moreover,

$$\|A\| = \mu_1 \; ; \quad \|A^{-1}\| = \mu_n^{-1} .$$

Finally, the orthogonal transformations do not change the norms of x and b.

We have

$$A^{-1} = \begin{bmatrix} \mu_1^{-1} & & & 0 \\ & \mu_2^{-1} & & \\ & & \ddots & \\ 0 & & & \mu_n^{-1} \end{bmatrix} .$$

If $b = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, and $db = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \epsilon \end{pmatrix}$,

then

$$x = A^{-1}b = \begin{pmatrix} \mu_1^{-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \text{and} \quad \delta x = A^{-1}\, db = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \epsilon\mu_n^{-1} \end{pmatrix} .$$

For these vectors,

$$\frac{\|\delta x\|}{\|x\|} = \frac{\epsilon\mu_1}{\mu_n} = \frac{\|db\|}{\|b\|}\,\frac{\mu_1}{\mu_n} = \mathrm{cond}(A)\,\frac{\|db\|}{\|b\|} \quad .$$

The last line shows that (5) is an equality in this case, as we promised to prove.

Although (5) is only an exact equality under exceptional conditions, it is usually rather close to equality, and in the following we assume approximate equality.

If $\mathrm{cond}(A) = 10^p$, and if b is known to be correct only to 10 decimals, then x can be known only to $10 - p$ decimals. Now p can range anywhere from 0 to ∞. The only hope of having any significance to x in a 10-decimal computing system is that, roughly,

$$\mathrm{cond}(A) \cdot 10^{-10} < \frac{1}{2} \quad .$$

In a base-β computer with t significant digits, we roughly need

$$\mathrm{cond}(A)\ \beta^{-t} < \frac{1}{2}$$

in order to have any significance to a solution.

Remember that all statements in this section are independent of any method of solving a system $Ax = b$. They are statements about errors in x which are inherent in the uncertainty in the data.

If A is subject to a change dA, and b is known exactly, then an inequality analogous to (5) is the following:

$$(6) \qquad \frac{\|\delta x\|}{\|x + \delta x\|} \leq \mathrm{cond}(A) \cdot \frac{\|dA\|}{\|A\|} \quad .$$

If $\|\delta x\|$ is small, compared with $\|x\|$, then we may safely consider the left-hand side of (6) as a relative error in x.

8. Accuracy achievable with Gaussian elimination. I assume that the reader knows what Gaussian elimination is, as a method of solving linear equation systems. The main strategic decision facing the designer of the algorithm is the choice of a unique pivot element for each of the n-1 stages in which a variable is eliminated from the remaining equations. There are two main strategies discussed:

(i) complete pivoting, in which at each stage one selects as a pivot some element $a_{i,j}$ of maximum absolute value among all the remaining elements of the matrix.

(ii) partial pivoting, in which at each stage one selects as a pivot some element $a_{i,j}$ of maximum absolute value among the first column of the remaining elements of the matrix.

Thus, in the first stage complete pivoting would search the whole matrix A for an element maximal in absolute value, whereas partial pivoting would search only the first column.

Some special classes of matrices permit elimination to proceed successfully without any search for pivoting--for example, positive definite symmetric matrices. But generally, pivotal searching is essential to guarantee success. The following simple example illustrates the disaster possible in not searching for a pivot. Consider a 3-digit floating-decimal machine.

The system is

$$\begin{cases} .0001\,x + 1.00\,y = 1.00 \\ 1.00\,x + 1.00\,y = 2.00 \end{cases}.$$

The true solution, rounded to five decimals, is $x = 1.00010$, $y = .99990$.

If one accepts the element .0001 as a pivot, the elimination of x from the second equation yields the equation

$$-10000\,y = -10000.$$

Backsolving, we find that $y = 1.00$, whence $x = \underline{0.00}$, a clear disaster.

On the other hand, partial pivoting would select the element $a_{2,1} = 1.00$ as the pivot. Elimination of x from the first equation yields the equation

$$1.00\ y = 1.00\ .$$

Backsolving, we get $y = 1.00$ and then $x = \underline{1.00}$, with obvious success.

We shall now assume that we are dealing with a t-digit base-2 floating-point computer. Rather than discuss the solution of a linear system, we shall consider the computation of the inverse A^{-1} of a given matrix. We wish to state the rounding error bounds that have been proved for Gaussian elimination.

Wilkinson [58] assumes a complete pivotal strategy, and that the matrix A is reasonably scaled at the start and at all intermediate stages (see Sec. 10 for more about scaling). Then, if all $|a_{i,j}| \leq 1$, a certain Gaussian algorithm yields a matrix X such that

$$\frac{\|X - A^{-1}\|}{\|A^{-1}\|} \leq (0.8)\, 2^{-t}\ n^{7/2}\ g(n)\ \|A^{-1}\| \quad . \tag{7}$$

Here $g(n)$ is the maximum of all elements of the successive matrices found during the elimination.

To express the result (7) in a form to be compared with those of Sec. 7, we note that $1 \leq \|A\| \leq n$, so that we expect that $\|A\| \doteq n^{1/2}$. Then we have roughly

$$\frac{\|X - A^{-1}\|}{\|A^{-1}\|} \leq n^3 \cdot 2^{-t}\ \mathrm{cond}(A)\ g(n) \quad . \tag{8}$$

What kind of bound can we give for $g(n)$? This turns out to be an open question. The best known result is approximately

$$g(n) \leq 1.8\ n^{(1/4)\log n}. \tag{9}$$

On the other hand, for all real matrices ever examined it has always been observed that

$$g(n) \leq n \ .$$

The last bound is attained for unboundedly large n by matrices related to the Hadamard matrices. For most matrices one even observes that

$$g(n) \leq 8 \ . \tag{10}$$

Tornheim [54] has found complex matrices A of unboundedly large n for which

$$g(n) \doteq 3.1\, n \ .$$

It would be most desirable to have a good bound for $g(n)$, so that (7) could be turned into a good a priori error bound for the computation of A^{-1}.

Naturally, for any particular matrix A, $g(n)$ is easily observed in the course of the elimination, so that in any event (7) becomes an a posteriori error bound. However, still better error bounds can be given a posteriori, as will be shown in Sec. 9.

Wilkinson's proof of (7) in [58] is reasonably short. It makes use of inverse rounding error analysis, which we shall mention again in Sec. 11. It is instructive to compare (8) with (6), even though one deals with inverses and one with linear systems. The factor 2^{-t} is essentially the inherent uncertainty level of the data, and should be equated to $\|dA\|/\|A\|$. Then the bound in (8) is larger than that in (6) by the factor $n^3\, g(n)$. Taking into account the empirical result (10) that $g(n) \leq 8$ for most real matrices, we then interpret (8) as saying that the computed matrix X generally differs from the true inverse A^{-1} in relative terms by no more than $8n^3$ times the error inherent in the problem. Thus simple Gaussian elimination is reasonably good at keeping the rounding error bound under control, for modest values of n. Much better results

can be achieved with some devices to be mentioned in Sec. 9.

The bound corresponding to (7) given by von Neumann and Goldstine [41] was

$$\frac{\|X - A^{-1}\|}{\|A^{-1}\|} \leq (5.3 + 14.6\ \|A\|^2)\ 2^{-t}\ n^2\ \|A^{-1}\|^2 \tag{11}$$

$$\sim\ 15\ n^2\ 2^{-t}[\mathrm{cond}(A)]^2 \quad .$$

The factor $[\mathrm{cond}(A)]^2$ arose from solving $A^TA\ x = A^Tb$, rather than $Ax = b$. The proof of (11) was an order of magnitude more difficult and tedious than the proof of (7).

9. More accurate solutions. Suppose that A is given as single-precision data, and that we wish to get solutions guaranteed to be more accurate than the above bounds would indicate. How shall we proceed? The most obvious choice is to perform all calculations in double-precision. Roughly speaking, then t is replaced by $2t$ in the above error bounds, and, since 2^{-2t} is so very much smaller than 2^{-t}, we gain many orders of magnitude in accuracy. The cost in computing time varies among different machines, but is only a factor of four on the IBM 7094. The cost in storage is greater, since we must double the storage reserved for the developing matrix.

Where the time and storage costs are too high to justify complete double precision, it is possible to make a very substantial gain by a much more limited use of double precision. Most of the operations in Gaussian elimination can be phrased as inner products of vectors of single-precision numbers. On many machines it is possible to accumulate such an inner product in double precision, and then round it off to single precision before storing away the result. The result of this accumulation is to reduce the maximum rounding error of an

inner product by a factor of n. The total effect turns out to be to reduce the round-off error bound by a factor of $n^{5/2}$. Thus, instead of the result (7), an elimination with pivoting and accumulation produces an approximate inverse X such that

$$\frac{\|X - A^{-1}\|}{\|A^{-1}\|} \leq 3.3n\, 2^{-t}\, \|A^{-1}\|\, g(n) \quad , \tag{12}$$

under certain additional hypotheses. See Wilkinson [61, p. 253]. The gain of the factor $n^{5/2}$ is very substantial, although experience shows that the actual errors in single-precision computation are usually rather less than the bounds.

One theoretical disadvantage of the complete pivoting strategy is that it does not mix well with the accumulation of inner products. When products are accumulated, one almost always uses a partial pivotal strategy, and accepts the theoretical possibility that pivots can grow very large.

A third and the most successful approach to increasing the accuracy of solutions of dense, stored linear systems is the so-called method of iterative improvement. By this method, if the matrix A is not too ill-conditioned, one in practice gets solutions which are the correctly rounded approximations to the true answers. We will now describe this development.

Suppose that by Gaussian elimination one has achieved a first approximate solution x_0 of the linear system $Ax = b$. The next step is to form the residual vector $r_0 = b - Ax_0$. If x_0 were the exact solution of the system, we would have $r_0 = \theta$, the null vector. If not, we solve the new linear system $Ay = r_0$, to obtain a vector y_1. Let $x_1 = x_0 + y_1$.

The process is repeated iteratively. I.e., for $k = 0, 1, 2, \ldots$ we form the residual $r_k = b - Ax_k$, solve the system $Ay = r_k$ to obtain a vector y_{k+1}, and then form $x_{k+1} = x_k + y_{k+1}$.

Under suitable hypotheses to be specified below, the sequence x_k converges to the true solution $A^{-1}b$ of the system $Ax = b$.

Several matters need to be clarified in this algorithm. First, it appears to involve a great deal of work to solve systems of the form $Ay = r_k$ for many values of k. In fact, this is not so. Gaussian elimination to solve a system $Ax = b$ involves three distinguishable stages:

(i) Triangularization of the matrix A by elementary row transformations

(ii) Application of the same row transformations to the right-hand side b

(iii) Solution of the triangular system by back-substitution.

It turns out that stage (i) requires approximately $n^3/3$ multiplications and additions, but that stages (ii) and (iii) together require only approximately n^2 multiplications and additions. Stage (i) need be done only once for all the systems $Ay = r_k$. If the multipliers defining the row transformations are saved, stages (ii) and (iii) can be done rapidly for each new system $Ay = r_k$ in turn. As a result, it is found that a sufficiently long sequence of vectors x_k can usually be computed in something like only 20 per cent more time than the computation of the first solution x_0.

It is absolutely essential that each residual vector r_k be computed to high precision. This is normally done by a double-precision accumulation of inner products, followed by rounding of the answer to single-precision floating-point form. If r_k is computed with merely a single-precision inner product, it will have rounding errors of several units in the least significant digits of x_k. Then the inequality (5), which is an approximate equality in practice, tells us that x_k will be wrong by several times $\mathrm{cond}(A)$ in its least significant digit. Since $\mathrm{cond}(A)$ may well be 10^4 or 10^5, the resultant accuracy in x_k is very low and, in fact, x_0 is itself almost as accurate as any succeeding x_k.

The following theorem gives the basis of the above method of iterative improvement:

Theorem. Let the matrix A have the property that

$$(0.8)\, 2^{-t}\, n^{7/2}\, g(n)\, \|A^{-1}\| < \frac{1}{2} \qquad (13)$$

Let the above algorithm be carried out, with each system $Ay = r_k$ being solved in single-precision base-2 floating-point arithmetic, but with computations of $r_k = b - Ax_k$ and $x_{k+1} = x_k + y_{k+1}$ carried out without rounding error. Then

$$\|x_k - A^{-1}b\| \to 0, \quad \text{as} \quad k \to \infty .$$

If the solution of the systems $Ay = r_k$ were done with accumulations of inner-products in double precision, then the left-hand side of (13) could be replaced by the right-hand side of (12)

In practice, of course, r_k is computed by a double-precision accumulation of inner products, and x_{k+1} is computed as the floating-point sum of x_k and y_{k+1} As a result, the sequence x_k does not converge to $A^{-1}b$ in the mathematical sense. Instead, x_k is observed to become constant at a value which is normally the correctly rounded single-precision approximation to $A^{-1}b$.

In the actual use of iterative improvement, one does not usually know in advance whether or not hypothesis (13) is satisfied, and it cannot be concluded afterwards either. Normal practice is therefore to rely on the following heuristic result:

Almost-theorem. Let the above algorithm be carried out, with each system $Ay = r_k$ being solved by the same version of Gaussian elimination, with each r_k being computed by a double-precision accumulation of inner products, and with x_{k+1} being computed as the floating-point sum of x_k and y_{k+1}.

If for $k \geq k_0$, all vectors x_k are equal to some single-precision vector x^*, then x^* is the correctly rounded single-precision approximation to $A^{-1}b$.

This almost-theorem cannot be proved, and, indeed, Kahan [32] has an extremely ingenious counter-example. However, most computers would bet their life on the applicability of the above almost-theorem in any practical example, unless Kahan were furnishing the problem!

Normally, when cond(A) gets near 2^t, the vectors x_k obviously diverge. Then there is no cure except to increase the precision with which the elimination is carried out, unless scaling **A** will help.

The usual value of k_0 is 3 or 4.

10. Scaling of matrices. One matter that was glossed over in Sec. 8 was the scaling of the matrix A before solving a system Ax = b. Alternate terms for scaling are preconditioning and equilibration. Suppose that the 2-by-2 numerical example of Sec. 8 were altered by multiplying the first equation by 10^6. Then the system would be

$$(14) \qquad \begin{cases} 10.0 \quad x + 100000 \quad y = 100000 \\ 1.00 \quad x + 1.00 \quad y = 2.00 \end{cases} .$$

The effect of the scaling is to make 10.0 the larger pivot in the first column. Then elimination of x from the second equation of (14) in 3-digit floating-decimal arithmetic will result in a new second equation

$$-10000\ y = -10000 .$$

Back solution leads to y = 1.00 and the awful result x = 0.00.

We see that poor scaling with a good pivotal strategy forces us into the same enormous rounding error that we obtained in Sec. 8 from the original set of equations and a bad pivotal strategy.

The conclusion of this is that a good pivotal strategy is only good when the matrix is properly scaled in advance. However, it must be admitted that so far we do not know guaranteed algorithms for scaling matrices well.

It is normal to scale matrices by simply multiplying rows and column by factors. In effect, one chooses nonsingular diagonal matrices D_1 and D_2, and then scales A by the transformation

$$A \rightarrow D_1^{-1} A D_2 .$$

Because $\text{cond}(A)$ is an ingredient of all our error bounds and convergence theorems, it is natural to wish to select D_1 and D_2 so as to reduce $\text{cond}(D_1^{-1} A D_2)$ to as low a value as is reasonably possible.

One usually uses powers of the floating-point base for scale factors, to avoid the introduction of rounding errors in the scaling. Or, alternatively, one may use the scaling only implicitly, without actually altering the elements of A.

Theorem (F. L. Bauer) <u>If the ordered set of pivotal elements is selected in advance, scaling of a matrix</u> A <u>by powers of the floating-point base does not change a single digit of the significand of any intermediate or final number in the solution of</u> $Ax = b$ <u>by Gaussian elimination.</u>

The theorem was presented in Bauer [1]. Thus the only possible effect of the scaling of A on the rounding errors must occur through changing the order of pivots. Our example showed that the change in pivots can indeed make a great deal of difference.

One is sometimes advised to pick D_1 and D_2 so that the resulting

matrix $D_1^{-1} A D_2$ has its maximum element in each row and column (in absolute value) in the interval $[.1, 1)$, in whatever number base one is using. However, Richard Hamming has showed (unpublished) that this advice does not always lead to good scaling. If

$$A = \begin{bmatrix} 1 & 1 & 2 \times 10^9 \\ 2 & -1 & 10^9 \\ 1 & 2 & 0 \end{bmatrix}$$

Then both of the following matrices are decimally scaled equivalents of A:

$$A_C = \begin{bmatrix} .1 & .1 & .2 \\ .2 & -.1 & .1 \\ .1 & .2 & 0 \end{bmatrix};$$

$$A_R = \begin{bmatrix} 10^{-10} & 10^{-10} & .2 \\ 2 \times 10^{-10} & -10^{-10} & .1 \\ .1 & .2 & 0 \end{bmatrix}.$$

However, A_C is a well-conditioned matrix that offers no difficulties in the solution of an equation system, whereas A_R is most ill-conditioned and provides vast troubles for elimination.

Bauer [2] has studied the problem of finding D_1 and D_2 to minimize $\mathrm{cond}(D_1^{-1} A D_2)$. It turns out that the solution depends on certain properties of the nonnegative matrices $|A| \cdot |A^{-1}|$ and $|A^{-1}| \cdot |A|$. (Here $|B|$ denotes the matrix of absolute values $|b_{i,j}|$.) Clearly, we can hardly hope to compute A^{-1} in order to find a reasonable scaling, so that we can compute A^{-1}!

So, it is an open question, how to find a demonstrably good and convenient scaling algorithm. Existing algorithms are either very superficial or potentially very slow.

The only cheerful side of the scaling question is that it seems to be a rare matrix which good scaling changes from untractable to tractable!

11. Analysis of rounding errors. We pointed out in Sec. 5 that the direct rounding error analysis of von Neumann and Goldstine was extremely tedious to apply. Givens [17] introduced the idea of inverse rounding errors. Wilkinson has developed this into a very powerful tool for bounding the rounding errors in matrix computations. The error bounds of Secs. 8 and 9 were obtained from inverse analysis. The basic idea is to change the nonassociative, non-distributive floating-point arithmetic system into an associative, distributive number system, by throwing the errors back onto the data of the computation.

For example, let $fl(u \times v)$ stand for the floating-point product (number base β) of the floating-point numbers u and v. The direct error analysis uses statements of the form

$$w = fl(u \times v) = uv + e, \quad \text{where} \quad |e| < \tfrac{1}{2} |uv| \beta^{1-t}$$

Further operations on w introduce new errors, and one has to keep account of the cumulation of all the old and new rounding errors. Eventually, one bounds the difference between the computed final answer and the mathematically correct answer corresponding to the given data.

In inverse analysis, one makes statements of the form

$$w = fl(u \times v) = uv(1 + \delta), \quad \text{where} \quad |\delta| < \tfrac{1}{2} \beta^{1-t} \quad .$$

Thus the computed product is considered the true mathematical product of (for example) the real numbers u and $v(1 + \delta)$, which differ slightly from u and v. Further floating-point operations on w produce numbers which are always treated as the results of exact operations on other slightly more perturbed approximations to the original data. The final answer is considered as the exact solution of an original problem with data which are perturbed by amounts for which bounds are given.

If desired, these inverse error bounds can be converted to ordinary error bounds, by normal mathematical methods.

Inverse error analysis turns out to be extremely well adapted to the analysis of algorithms of a marching type which continally introduce new data. Both the solution of linear equations and the evaluation of polynomials are of this type. Inverse error analysis is not at all well suited for problems of an iterative nature--for example, the Newton process for evaluating the square root of a number.

The reader is referred to Wilkinson [60, 62] for further study of inverse round-off analysis.

A second approach to round-off analysis is the interval analysis, extensively developed by Moore [40], but based on the idea of "range numbers" presented earlier by Dwyer [6]. In its original form, interval analysis is poorly adapted to matrix computations, but Hansen [23] has modified it ingeniously for matrix work.

12. Eigenvalues of symmetric matrices. Space does not permit as extensive a treatment of the eigenvalue problem as that given for the linear equations problem. We can only mention a few highlights of today's methods. The reader is referred to Wilkinson's treatise [61] for an almost complete presentation of the state of the art.

As with the linear equations problem, the computation of eigenvalues of matrices divides into two methods, according to the nature of the matrices. For large, sparse matrices the methods are mostly infinite iterations, and will not be considered here. For dense, stored matrices, most methods are finite algorithms.

If a matrix A is symmetric, its eigenvalues are very well determined by the data. In fact, let the symmetric matrix $B = A + E$ have eigenvalues β_i, and let A (also symmetric) have eigenvalues α_i. Then the eigenvalues can be so numbered that

$$|\alpha_i - \beta_j| \leq \|E\| , \text{ for all } i. \tag{15}$$

Now inverse error analysis refers the computed eigenvalues of a matrix A back to a matrix $B = A + E$. If E can be proved to be small (as it can), then (15) shows how small the eigenvalue errors are. In fact, today's methods can yield eigenvalues that are in error by only a few digits in the least significant digits of the large eigenvalues.

The method of Jacobi [31] is an infinite iteration for dense, stored matrices. It produces a sequence of matrices orthogonally congruent to A:

$$A_k = U_k^T A U_k .$$

Moreover, A_k converges to a diagonal matrix D whose diagonal entries are, of

course, the eigenvalues of A. In fact, each A_{k+1} is computed from the previous A_k by a rotation in the coordinate 2-space of some two indices i and j, a rotation chosen so that $a_{i,j}^{(k+1)} = 0$.

For any k such that A_k is almost diagonal, the columns of the corresponding orthogonal matrix U_k are approximately column eigenvectors of A. Moreover, the columns are themselves orthogonal. Thus the Jacobi method yields approximate eigenvectors of fine quality as a by-product of the basic iteration. The whole program is easy to write, and it is difficult for it to be done badly. There are some theoretical problems about how good the eigenvectors are, and whether the U_k actually converge.

Goldstine, Murray, and von Neumann [19] analyzed the rounding errors in a fixed-point version of the Jacobi method.

The original Jacobi algorithm chose i and j to maximize the absolute value of the element $a_{i,j}^{(k)}$ of A_k. Modern algorithms modify this criterion in one of two ways:

(i) In the cyclic Jacobi methods, the off-diagonal elements $a_{i,j}$ are zeroed in some cyclic order. Forsythe and Henrici [11] proved the convergence of a common cyclic method. See also Hansen [22].

(ii) In threshold Jacobi methods, an element $a_{i,j}$ is selected for annihilation only when its absolute value is above a certain threshold size, which gets smaller as the iteration progresses. See Pope and Tompkins [49] and Corneil [4].

It has been proved only recently that the cyclic Jacobi method converges quadratically for any matrix A. See Schönhage [52] and Wilkinson [59]. The work was based on that of Henrici [24].

Givens [17] observed that, although it takes an infinite sequence of

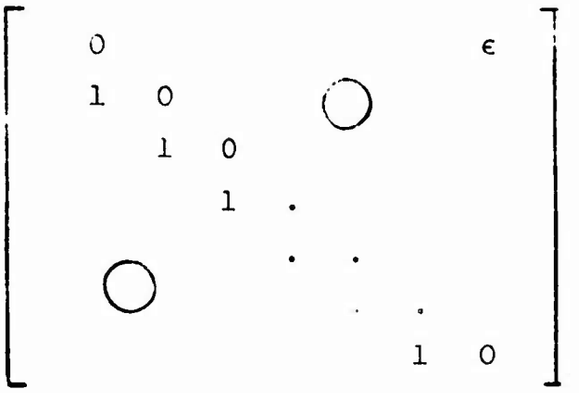
rotations to bring A_k to diagonal form, a mere $\frac{1}{2}(n-1)(n-2)$ rotations can bring A_k to tridiagonal form. This reduced the problem to that of finding eigenvalues of tridiagonal matrices, and the latter problem has been a subject of research ever since. See Ortega [43], Ortega and Kaiser [45], and recent work of Kahan and Varah [33]. In any case, the Givens idea cut the practical time of finding eigenvalues by a factor of about 9 in practice (Wilkinson [61, p. 335]). A few years later Householder (see Householder and Bauer [29]) introduced a new method of tridiagonalizing a symmetric matrix, using n-2 reflections instead of $\frac{1}{2}(n-1)(n-2)$ rotations. This cut the time down by another factor of two, and effectively put the Givens method out of business. An error analysis is given by Ortega [44]. Most contemporary programs use the Householder method, but differ widely in how eigenvalues of tridiagonal matrices are found. Getting the eigenvectors is surprisingly tricky, and lack of knowledge of how to do it is one reason for the occasional continued use of the Jacobi methods.

13. Eigenvalues of unsymmetric matrices. The area of greatest activity in the past decade of research on computational linear algebra has been the eigenvalue problem for unsymmetric matrices. Only one method from before the computer era is still in use--the power method--and it has only limited applications today. Most methods in use today were unheard of 15 years ago.

It is essential to realize the instability inherent in the eigenvalue problem for unsymmetric matrices. In contrast to the close bound (15), for unsymmetric matrices the corresponding result, due to Ostrowski [46], is

$$|\alpha_i - \beta_i| \leq (\text{polynomial in } n) \times \|E\|^{1/n} \quad (\text{all } i) . \tag{16}$$

The above result is very weak, and yet is the best possible general result of its kind. For the matrix



of order n has all eigenvalues 0 for $\epsilon = 0$, but all eigenvalues are distinct and have modulus $|\epsilon|^{1/n}$ for $\epsilon \neq 0$. Thus, if $n = 40$ and $\epsilon = 10^{-40}$, all eigenvalues have modulus 0.1 (!).

Fortunately, eigenvalues are not usually so sensitive. In fact, different eigenvalues of a matrix A can differ enormously in their sensitivity to perturbations in A. Chapter 2 of Wilkinson [61] is full of useful results. They are generally a posteriori results, giving bound for the changes in eigenvalues as functions of perturbations in a matrix and information about the other eigenvalues and eigenvectors.

The great power of the Jacobi method for symmetric matrices, and the extremely pleasant rounding characteristics of unitary matrices led to a desire to use them for the unsymmetric eigenvalue problem. The basic theorem is due to Schur:

Theorem. For an arbitrary matrix A, there exists a unitary matrix U such that

$$T = U^H A U$$

is triangular. (Here U^H denotes the conjugate transpose of U.)

Since the eigenvalues of A are the diagonal elements of T, the hope has been to find unitary matrices which bring A nearly into a triangular form, and then let the diagonal elements serve as approximate eigenvalues of A.

Investigations by Greenstadt [21], Lotkin [37], and Eberlein [7] offer some hope, but no real promise of success

For most methods of attacking the eigenvalue problem, the first step is to condense the data, to save time and storage in further work. The now universally accepted condensed form is the Hessenberg matrix, in which $a_{i,j} = 0$ for $i - j > 1$ (or its transpose) It is possible to transform A by orthogonal congruences of the Householder type into a Hessenberg form with only very small rounding errors. Any further condensation (say, into tridiagonal form) is subject to serious losses of digits. A transformation to the companion-matrix form is particularly disastrous in practice, and it normally requires very substantial increases in precision to successfully yield the eigenvalues of A.

As an alternative, one can transform A to Hessenberg form by Gaussian elimination with partial pivoting, a similarity transformation

The next stage in the unsymmetric eigenvalue problem is to get the eigenvalues of a Hessenberg matrix H. A variety of methods have been used.

(i) One can search for zeros of $\det(H - zI)$ by root-finding methods, for complex z. The most satisfactory method appears to be that proposed by Hyman [30] and developed by Parlett [47] in a number of programs. He makes use of the method of Laguerre to find the zeros of $f(z)$, following by a form of zero suppression Very satisfactory recurrences are used to evaluate $f(z)$, $f'(z)$, and $f''(z)$, as needed by the Laguerre process. After the eigenvalues $\lambda_1, \ldots, \lambda_r$ have been found, they are suppressed by applying the Laguerre process to

$$f(z) / \prod_{i=1}^{r} (\lambda - \lambda_i) .$$

(ii) The LR-algorithm of Rutishauser [58] was an important development. Since it has now been pretty well supplanted by the QR-algorithm, we omit mention of it.

(iii) Francis [15, 16] in England, and Kublanovskaja [35] in the Soviet Union devised the very interesting QR-algorithm. This is now widely considered the most satisfactory eigenvalue algorithm for dense, stored unsymmetric matrices.

The basic theorem is that an arbitrary real square matrix A can be factored in the form $A = QR$, where Q is orthogonal, and where R is an upper-triangular matrix with all diagonal elements $r_{i,i}$ nonnegative. If A is nonsingular, then both Q and R are unique.

In fact, the computation is done by building up an orthogonal matrix Q^T such that $Q^TA = R$, where R has the above properties.

As an aside, for nonsingular A, the reader will be more familiar with the stepwise determination of an upper-triangular matrix R with positive $r_{1,1}$ such that AR^{-1} is an orthogonal matrix Q. This is the matrix expression of the familiar Gram-Schmidt process of analysis. It will perhaps surprise the reader that the matrix Q resulting from the Gram-Schmidt algorithm is normally far from orthogonal, because of rounding errors. On the other hand, if the same Q is determined so that $Q^TA = R$, the rounding errors are very small.

The basic QR-algorithm proceeds as follows. Let $H = H_0$ be a Hessenberg matrix. For $k = 0, 1, 2, \ldots,$ factor H_k in the form

$$H_k = Q_k R_k \quad ,$$

and then form

$$H_{k+1} = R_k Q_k \quad .$$

It is easily shown that H_{k+1} is also a Hessenberg matrix.

The basic theorem is the following.

Theorem. Let H have eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ with

$$|\lambda_1| < |\lambda_2| < \ldots < |\lambda_n| \quad . \tag{17}$$

Then the matrices H_k converge to an upper-triangular matrix whose diagonal elements are the eigenvalues of H

In the more usual case where (17) is not satisfied, we find that H_k converges in shape to a blockwise triangular matrix. (This means that outside a blockwise triangular matrix all elements of H_k tend to zero, as $k \to \infty$, but that some elements of the blockwise triangular form may not converge.) Moreover the 2-by-2 and 1-by-1 diagonal blocks of the matrix H_k have eigenvalues which in their totality converge to the eigenvalues of H.

For simplicity, consider a matrix H with eigenvalues

$$0 < \lambda_1 < \lambda_2 < \cdots < \lambda_n .$$

The QR method for such a matrix converges with an error which is

$$O[(\lambda_1/\lambda_2)^k] .$$

The convergence would be more rapid if, instead of H, we dealt with the matrix $H - pI$, where $0 < p < \lambda_1$. If p were practically equal to λ_1, the convergence would be extremely rapid. Modifications of the QR-algorithm have been devised that simulate this so-called origin shift which introduces a p near λ_1. After one eigenvalue λ_1 has been isolated, the QR method can then be applied to an $n-1$ by $n-1$ matrix with eigenvalues $\lambda_2, \ldots, \lambda_n$. New origin shifts are then introduced to bring out λ_2 as rapidly as possible. With well devised origin shifts, the whole process has been observed to converge with an average of less than two iterative steps per eigenvalue.

Most research goes into the invention of origin shifts when some of the eigenvalues are complex and of equal modulus. We shall not attempt to give the ideas.

A more recent convergence proof has been given by Wilkinson [62], but, like the Francis proof, is given for an arbitrary matrix A. If one limits himself to matrices of Hessenberg form, easier proofs can be given; see Kahan (unpublished).

Normally, the eigenvalues are obtained in order of increasing modulus. Parlett [48] has given theorems stating precisely when this occurs.

If H is a symmetric band matrix, then the QR-algorithm preserves the band width during the iteration, and is very satisfactory. In particular, QR is a possible algorithm for computing eigenvalues of a symmetric tridiagonal matrix.

If H is an unsymmetric band matrix, the QR-algorithm loses the zero bands above the diagonal.

So far, we have not mentioned getting the eigenvectors of a Hessenberg matrix H. This is the most difficult problem we shall mention. The prevailing method is that of inverse iteration. The eigenvalues are assumed already known. For any fixed eigenvalue λ, one selects a vector x_0 arbitrarily. Then one carries out an iteration of the following form:

For each $k = 0, 1, 2, \ldots,$ find x_{k+1} by solving the system

$$(H - \lambda I)x_{k+1} = x_k \quad . \tag{18}$$

One continues until x_k is quite large. In easy cases, x_k is nearly an eigenvector belonging to λ. Wilkinson [61, Chap. 9] discusses variants of this process. Varah [56] has written several algorithms

If H is a real Hessenberg matrix, but λ is a complex eigenvalue, one has to choose between doing complex arithmetic, or some judiciously selected process with real arithmetic.

Finally, one transforms x_k back to the original coordinate system of A by undoing the orthogonal transformations from A to H.

If some of the eigenvalues of H are very close, the real problems begin. A pair of close eigenvalues may in fortunate cases have distinct column eigenvectors that are far from parallel; this represents an approximation to a double eigenvalue

with a linear elementary divisor. It is far more likely that a pair of close eigenvalues will have column eigenvectors that are almost parallel. This represents an approximation to the infinitely more probable case of a double eigenvalue with nonlinear divisors.

In the former case, it is not difficult to compute two eigenvectors that are far from parallel. It is only necessary to carry out the iteration (18) with different x_0, or with two slightly different values of λ.

In the latter case, it appears difficult to obtain much from the iteration but a single eigenvector belonging to λ. What should be done next? In part one doesn't know what the problem proposer would like. In part one doesn't know what is possible. Varah is carrying out research on the problem. He is attempting to find an orthogonal basis for the invariant subspace of dimension 2 (in this case) belonging to λ.

For a "nice" matrix, Varah is also getting guaranteed error bounds for all eigenvalues and all eigenvectors, using Gerschgorin theorems, as Wilkinson recommends.

14. Conclusion and moral. The computational methods of linear algebra are moving into a stage where we have reasonably satisfactory methods for dense, stored matrices A. The main exception is the problem of getting eigenvectors with error bounds, for unsymmetric matrices. The algorithms have been refined several times, and are being published, particularly in Numerische Mathematik. Casual users of matrix algebra will do no better than to take such algorithms "off the shelf" for their problems. The best algorithms are mainly written in Algol 60. Even though the reader may use another language, it is unquestionably worthwhile for him to learn to read Algol 60, just in order to be able to read these algorithms and adapt them to his own problems.

No method of solving a computational problem is really available to a user until it is completely described in an algebraic computing language and made completely reliable. Before that, there are indeterminate aspects in every algorithm. Frequently the entire advantage of a certain computing process lies in the treatment of certain fine points which can hardly be suspected until they are completely programmed. This is the reason why the amateur should either consult an expert, or take great pains to pick up a foolproof algorithm. This is the reason why professionals should concentrate very hard on completely foolproofing the algorithms they devise, before putting them on the shelf for widespread use.

REFERENCES

[1] F. L. Bauer, "Optimal scaling of matrices and the importance of minimal condition," pp. 198-201 of C. M. Popplewell (editor), Information Processing, North Holland Publishing Co., 1962.

[2] F. L. Bauer, "Optimally scaled matrices," Numerische Math., vol. 5 (1963), pp. 73-87.

[3] Richard Bellman, Introduction to Matrix Analysis, McGraw-Hill, 1960, 328 pp.

[4] Derek Corneil, Eigenvalues and Orthogonal Eigenvectors of Real Symmetric Matrices, Master's thesis, Department of Computer Science, University of Toronto, 1965, 78 pp.

[5] George B. Dantzig, Linear Programming and Extensions, Princeton Univ. Press, 1963, 625 pp.

[6] Paul S. Dwyer, Linear Computations, Wiley, 1951, 344 pp.

[7] P. J. Eberlein, "A Jacobi-like method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix," J. Soc. Indust. Appl. Math., vol. 10, pp. 74-88 and (errata) 393.

[8] D. K. Faddeev and V. N. Faddeeva, Computational Methods of Linear Algebra (translated by Robert C. Williams from Russian book of 1960), W. H. Freeman and Co., 1963, 620 pp.

[9] V. N. Faddeeva, Computational Methods of Linear Algebra (translated by Curtis D. Benster from Russian book of 1950), Dover, 1959, 252 pp.

[10] George E. Forsythe, "Solving linear algebraic equations can be interesting," Bull. Amer. Math. Soc., vol. 59 (1953), pp. 299-329.

[11] G. E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix," Trans. Amer. Math. Soc., vol. 94 (1960), pp. 1-23.

[12] George E. Forsythe and Cleve R. Moler, Computer Solution of Linear Algebraic Systems, to appear.

[13] George E. Forsythe and Wolfgang R. Wasow, Finite Difference Methods for Partial Differential Equations, Wiley, 1960, 444 pp.

[14] L. Fox, An Introduction to Numerical Linear Algebra, Clarendon Press. 1964, 328 pp.

[15] J. G. F. Francis, "The QR transformation; a unitary analogue to the LR transformation--part I," Computer J., vol. 4 (1961), pp. 265-271.

[16] J. G. F. Francis, "The QR transformation--part II," Computer J., vol 4 (1962), pp. 332-345.

[17] Wallace Givens, "Numerical Computation of the Characteristic Values of a Real Symmetric Matrix," Report ORNL 1574, Oak Ridge National Laboratory, Oak Ridge, Tenn., 1954, 107 pp.

[18] G. Golub and W. Kahan, "Calculating the singular values and pseudoinverse of a matrix," J. SIAM Numer. Anal. Ser. B, vol 2 (1965), pp. 205-224

[19] H. H. Goldstine, F. J. Murray, and J. von Neumann, "The Jacobi method for real symmetric matrices," J. Assoc. Comput. Mach., vol. 6 (1959), pp. 59-96.

[20] Herman H. Goldstine and John von Neumann, "Numerical inverting of matrices of high order. II," Proc. Amer. Math. Soc., vol. 2 (1951), pp. 188-202. (See first article listed under von Neumann.)

[21] J. Greenstadt, "A method for finding roots of arbitrary matrices," Math Tables Other Aids Comput., vol. 9 (1955), pp 47-52.

[22] Eldon R. Hansen, "On cyclic Jacobi methods," J. Soc. Indust. Appl. Math., vol. 11 (1963), 448-459.

[23] Eldon Hansen, "Interval arithmetic in matrix computations, part I," J. SIAM Numer. Anal. Ser. B, vol. 2 (1965), pp. 308-320.

[24] Peter Henrici, "On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing eigenvalues of Hermitian matrices," J. Soc. Indust. Appl. Math., vol. 6 (1958), pp 144-162.

[25] Peter Henrici, "The quotient-difference algorithm," Nat. Bur. Standards Appl. Math. Ser., vol. 49 (1958), pp. 23-46.

[26] Peter Henrici, L. Fox, and Cleve R. Moler, manuscript in preparation.

[27] Magnus R. Hestenes and Eduard Stiefel, "Methods of conjugate gradients for solving linear systems," J. Res. Nat. Bur. Standards, vol. 49 (1952), pp. 409-436

[28] Alston S. Householder, The Theory of Matrices in Numerical Analysis, Blaisdell Publ. Co., 1964, 257 pp.

[29] Alston S. Householder and Friedrich L. Bauer, "On certain methods for expanding the characteristic polynomial," Numerische Math., vol. 1 (1959) pp. 29-37.

[30] Morton A. Hyman, "Eigenvalues and eigenvectors of general matrices," paper presented to Twelfth National Meeting of Association for Computing Machinery, Houston, Texas, 1957.

[31] C. G. J. Jacobi, "Über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen," J. Reine Angew. Math., vol. 30 (1846), pp 51-95.

[32] W. Kahan, "Numerical linear algebra," Canadian Math. Bull., to appear

[33] W. Kahan and J. Varah, "Two working algorithms for the eigenvalues of a symmetric tridiagonal matrix," Report CS43, Computer Science Dept. Stanford University, 1966, 29 pp.

[34] V. V. Klyuyev and N. I. Kokovkin-Shcherbak, "On the minimization of the number of arithmetic operations for the solution of linear algebraic systems of equations" (translated by G. J. Tee from a Russian article of 1965), Technical Report CS24, Computer Science Department, Stanford University, 1965, 24 pp.

[35] V. N. Kublanovskaja, "O nekotoryh algorifmah dlja rešenija polnoĭ problemy sobstvennyh značeniĭ," Ž. Vyčis. Matem. Matem. Fiz., vol. 1 (1961), pp. 555-570.

[36] Cornelius Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," J. Res Nat. Bur. Standards, vol. 45 (1950), pp. 255-282.

[37] Mark Lotkin, "Characteristic values of arbitrary matrices," Quart. Appl. Math., vol. 14 (1956), pp. 267-275.

[38] Marvin Marcus and Henry Minc, A Survey of Matrix Theory and Matrix Inequalities, Allyn and Bacon, 1964, 180 pp.

[39] Cleve B. Moler, "Iterative refinement in floating point," J. Assoc. Comput. Mach., vol. 14 (1967), 000-000.

[40] Ramon E. Moore, "The automatic analysis and control of error in digital computing based on the use of interval numbers." pp. 61-130 of Louis B. Rall (editor), Error in Digital Computation, vol. 1, Wiley, 1965, 324 pp.

[41] John von Neumann and H. H. Goldstine, "Numerical inverting of matrices of high order," Bull. Amer. Math. Soc., vol. 53 (1947), pp. 1021-1099. (See sequel listed under Goldstine.)

[42] Ben Noble, Applied Linear Algebra, preliminary edition, Prentice Hall, 1966, 413 pp.

[43] J. M. Ortega, "On Sturm sequences for tridiagonal matrices," J. Assoc. Comput. Mach., vol. 7 (1960), pp. 260-263.

[44] James M. Ortega, "An error analysis of Householder's method for the symmetric eigenvalue problem," Numerische Math., vol. 5 (1963), pp. 211-225.

[45] James M. Ortega and Henry F. Kaiser, "The LL^T and QR methods for symmetric tridiagonal matrices," Computer J., vol. 6 (1963), pp. 99-101.

[46] Alexander M. Ostrowski, "Über die Stetigkeit von charakteristischen Wurzeln in Abhängigkeit von den Matrizenelementen," Jber. Deutsch. Math. Verein., vol. 60. Abt. 1 (1957), pp. 40-42.

[47] Beresford Parlett, "Laguerre's method applied to the matrix eigenvalue problem," Math. Comput., vol. 18 (1964), pp. 464-485.

[48] Beresford Parlett, "Convergence of the QR algorithm for Hessenberg matrices." to appear.

[49] David A. Pope and C. Tompkins, "Maximizing functions of rotations--experiments concerning speed of diagonalization of symmetric matrices using Jacobi's method," J. Assoc. Comput. Mach., vol. 4 (1957), pp. 459-466.

[50] Heinz Rutishauser, Der Quotienten-Differenzen-Algorithmus, Birkhäuser Verlag, 1957, 74 pp.

[51] Heinz Rutishauser, "Solution of eigenvalue problems with the LR-transformation," Nat. Bur. Standards Appl. Math. Ser., vol. 49 (1958), pp. 47-81.

[52] A. Schönhage, "Zur quadratischen Konvergenz des Jacobi-Verfahrens," Numerische Math., vol. 6 (1964), pp. 410-412.

[53] John Todd, "The condition of a certain matrix," Proc. Cambridge Philos. Soc., vol. 46 (1949), pp. 116-118.

[54] L. Tornheim, "Maximum third pivot for Gaussian reduction," manuscript, California Research Corporation, Richmond, Calif., 1965, 10 pp.

[55] A. M. Turing, "Rounding-off errors in matrix processes," Quart. J. Mech. Appl. Math., vol. 1 (1948), pp. 287-308.

[56] J. M. Varah, "Eigenvectors of a real matrix by inverse iteration," Technical report CS34, Computer Science Dept., Stanford University, 1966, 24 pp.

[57] Richard S. Varga, Matrix Iterative Analysis, Prentice-Hall, 1962, 322 pp.

[58] J. H. Wilkinson, "Error analysis of direct methods of matrix inversion," J. Assoc. Comput. Mach., vol. 8 (1961), pp. 281-330.

[59] J. H. Wilkinson, "Note on the quadratic convergence of the cyclic Jacobi process," Numerische Math., vol. 4 (1962), pp. 296-300.

[60] J. H. Wilkinson, Rounding Errors in Algebraic Processes, H. M. Stat. Office and Prentice Hall, 1963, 161 pp.

[61] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, 1965, 662 pp.

[62] J. H. Wilkinson, "Convergence of the LR, QR, and related algorithms," Computer J., vol. 8 (1965), pp. 77-84.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Science Department Stanford University Stanford, Calif. 94305	2a. REPORT SECURITY CLASSIFICATION Unclass. 2b. GROUP ---

3. REPORT TITLE

TODAY'S COMPUTATIONAL METHODS OF LINEAR ALGEBRA

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Manuscript for publication (Technical Report)

5. AUTHOR(S) (Last name, first name, initial)

FORSYTHE, George E.

6. REPORT DATE August 11, 1966	7a. TOTAL NO. OF PAGES 47	7b. NO. OF REFS 62
8a. CONTRACT OR GRANT NO. Nonr-225(37) b. PROJECT NO. NR-044-211 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) CS46 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) none	

10. AVAILABILITY/LIMITATION NOTICES

Releasable without limitations on dissemination.

11. SUPPLEMENTARY NOTES ---	12. SPONSORING MILITARY ACTIVITY Office of Naval Research, Code 432 Washington, D. C. 20360

13. ABSTRACT

This is a survey of selected computational aspects of linear algebra, addressed to the nonspecialist in numerical analysis. Some current methods of solving systems of linear equations, and computing eigenvalues of symmetric and unsymmetric matrices are outlined. There is a bibliography of 62 titles.

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
1. Eigenvalues. 2. Linear algebra. 3. Computational methods. 4. Matrix inversion. 5. Rounding error.						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.