Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store

By B. A. Chartres

The Jacobi method for diagonalizing symmetric matrices requires, in its usual form, that the matrix be held in a random-access store. Two variations of the method are described which make it suitable for matrices held on a serial-access store, such as a magnetic tape. Techniques are described for applying these methods to computers with from one to six magnetic-tape units.

1. Introduction

The Jacobi (Goldstine, Murray and von Neumann, 1959; Gregory, 1953; Householder, 1953; Pope and Tompkins, 1957; Henrici, 1958; Forsythe and Henrici, 1960) and Givens (Givens, 1957; Wilkinson, 1958) methods can well be described as the two most popular methods for the computation, on automatic digital computers, of the eigenvalues and eigenvectors of real symmetric matrices. The Jacobi iterative reduction to diagonal form, and the reduction to tri-diagonal form which forms the first stage of the Givens process, both involve the application to the symmetric matrix of orthogonal similarity transformations. Each individual transformation involves a re-calculation of all elements in two of the rows and the two corresponding columns of the matrix. This requires that the matrix be stored in a store which allows immediate access to the affected elements.

If the immediate-access working store of the computer is not large enough to hold the whole matrix, then it is necessary to store the matrix on a backing store, such as a magnetic tape, and to bring into the working store only that part of the matrix which is to be operated upon at any one time. The problem of obtaining access to the required rows and columns of the matrix then becomes very acute—there appears to be no way of arranging the matrix on a magnetic tape which will yield quick access to the elements in the sequence required for the conventional Jacobi and Givens methods.

Rollett and Wilkinson (1961) and Johansen (1961) have developed techniques for applying the Givens reduction to matrices stored row-by-row on magnetic tape. In both these techniques all the operations required to produce a complete column of zeros are performed during a single scan of the matrix, the order in which the individual operations are performed being determined by the sequence in which the matrix elements become available.

This paper describes an alternative solution applicable to both the Jacobi and Givens methods. This is the application of the very simple idea of replacing the "two-sided" similarity transformation* $\boldsymbol{B} = \boldsymbol{XAX}^T$ by the "one-sided" operation $\boldsymbol{B} = \boldsymbol{XA}$. Hence, instead of operating on both the rows and columns of the matrix $\boldsymbol{A}$ we operate on its rows only. This enables the calculation to be organized in such a way that the elements of $\boldsymbol{A}$ are operated upon in a systematic row-by-row order.

* The superfix T indicates the transpose of a matrix.

Two methods of applying the one-sided transformation are proposed. The first method involves a 37% increase in computation over the regular Jacobi method, while the second is 37% shorter. When applied to the Givens process the second method requires nearly twice as many arithmetic operations as the usual procedure, while the first method is twice as long again, whereas the techniques of Rollett, Wilkinson and Johansen do not involve any increase at all. Only the application to the Jacobi method will therefore be described here.*

The technique described here was developed for the computer SILLIAC which, at the time of writing, has a 1,024-word immediate-access store and a single magnetic-tape unit which allows the tape to be read and written while moving in either direction. Methods of applying the technique to computers with from one to six tape units that can read and write in the forward direction only are also described in this paper.

2. Modifying the Jacobi Method

In the Jacobi method the symmetric matrix $\boldsymbol{A}$ is converted to a diagonal matrix $\boldsymbol{D}$ by the transformation

$$\boldsymbol{WAW}^T = \boldsymbol{D} \tag{1}$$

where $\boldsymbol{W}$ is the limit, as r goes to infinity, of the product

$$\boldsymbol{W}_r = \boldsymbol{V}_r\boldsymbol{V}_{r-1} \dots \boldsymbol{V}_2\boldsymbol{V}_1 \tag{2}$$

of simple orthogonal matrices $\boldsymbol{V}_r$. Thus $\boldsymbol{D}$ is the limit of a sequence

$$\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3, \dots$$

where

$$\boldsymbol{A}_1 = \boldsymbol{A}$$

$$\boldsymbol{A}_{r+1} = \boldsymbol{V}_r\boldsymbol{A}_r\boldsymbol{V}_r^T \tag{3}$$

Each $\boldsymbol{V}_r$ is identical to the unit matrix except in the four elements

$$v_{pp} = v_{qq} = \cos\theta_r \tag{4}$$
$$v_{qp} = -v_{pq} = \sin\theta_r.$$

* A full description of the application of the technique described in this paper to the Givens method is available in Technical Report No. 8 of the Basser Computing Department, University of Sydney, Sydney; "Adaptation of the Jacobi and Givens methods for a Computer with Magnetic-tape Backing Store."

Hence the transformation defined in equation (3) has the effect of replacing both the pth and qth rows of $\boldsymbol{A}_r$ by linear combinations of the two rows, and similarly for the pth and qth columns. The pair (p, q) is chosen for each r either in a purely cyclic manner, or in a cyclic manner modified according to the relative magnitude of the off-diagonal elements of $\boldsymbol{A}_r$ (see Henrici, 1958). The angle θ_r is chosen in each case in such a way that the sequence of matrices $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3, \ldots$ converges to a diagonal matrix. Normally θ_r is chosen so that the (p, q) element of $\boldsymbol{A}_r$ is reduced to zero, this being the choice which gives a maximum reduction in the sum of squares of the off-diagonal elements.

For our purpose it is important to note that the multiplication of $\boldsymbol{A}_r$ on the left by $\boldsymbol{V}_r$ has the effect of modifying only the elements in the pth and qth rows of $\boldsymbol{A}_r$, while the multiplication on the right by $\boldsymbol{V}_r^T$ modifies only the elements in the pth and qth columns; it is this necessity to modify elements according to both row and column arrangement that makes it necessary in the standard Jacobi method to have effectively random access to the elements of $\boldsymbol{A}_r$.

We express equation (1) in the form

$$\boldsymbol{WA} = \boldsymbol{DW} \tag{1'}$$

preserving equations (2) and (4). The modified form (1′) suggests the modification to equation (3)

$$\boldsymbol{A}_{r+1} = \boldsymbol{V}_r\boldsymbol{A}_r \tag{3'}$$

If we denote the ith row of $\boldsymbol{A}_r$ by $a_i^{(r)}$ we have the transformation relations

$$\left.\begin{aligned} a_p^{(r+1)} &= a_p^{(r)}\cos\theta - a_q^{(r)}\sin\theta \\ a_q^{(r+1)} &= a_p^{(r)}\sin\theta + a_q^{(r)}\cos\theta \end{aligned}\right\} \tag{5}$$

with relations of identical form connecting the rows of $\boldsymbol{W}_r$ and $\boldsymbol{W}_{r+1}$.

Of course, the sequence of matrices $\boldsymbol{A}_r$, $r = 1, 2, \ldots$, no longer converges to a diagonal matrix. Rather it converges to a matrix which can be expressed in the form $\boldsymbol{DW}$ where $\boldsymbol{D}$ is diagonal and $\boldsymbol{W}$ is orthogonal, i.e. to a matrix whose rows form a non-normalized orthogonal set. We must, therefore, now choose the angles θ_r such that the sequence $\boldsymbol{A}_r$ converges to a matrix with orthogonal rows. There are two ways of doing this; one yields exactly the same set of angles θ_r as does the Jacobi method when applied to the same matrix, the other yields the set of angles that would be used if the Jacobi method were applied to the matrix A^2.

Method I

We compute the sequence of matrix pairs $(\boldsymbol{A}_r, \boldsymbol{W}_r)$, $r = 1, 2, \ldots$, defined by

$$\begin{aligned} \boldsymbol{A}_{r+1} &= \boldsymbol{V}_r\boldsymbol{A}_r \\ \boldsymbol{W}_{r+1} &= \boldsymbol{V}_r\boldsymbol{W}_r \end{aligned} \tag{6}$$

with $\boldsymbol{A}_1 = \boldsymbol{A}$, $\boldsymbol{W}_1 = \boldsymbol{I}$. This calculation is performed by constructing the combined matrix $(\boldsymbol{A}, \boldsymbol{I})$ and performing the operations on it.

Then, at every value of r, the product

$$\boldsymbol{A}_r\boldsymbol{W}_r^T = \boldsymbol{W}_r\boldsymbol{A}\boldsymbol{W}_r^T \tag{7}$$

is a symmetric matrix similar to $\boldsymbol{A}$. Consequently, if we choose the angles θ_r exactly as in the Jacobi process, we are carrying out what is essentially the same transformations on $\boldsymbol{A}$, except that we are keeping the transformed $\boldsymbol{A}$ in a factored form.

Now the calculation of θ_r involves the (p, p), (q, q) and (p, q) elements of $\boldsymbol{W}_r\boldsymbol{A}\boldsymbol{W}_r^T$. In the regular Jacobi method these are immediately available, but in our modified method they must be computed as scalar products of the pth and qth rows of $\boldsymbol{A}_r$ and $\boldsymbol{W}_r$. This does not affect the problem of obtaining access to the matrix elements, for the rows whose scalar products are required are the very rows upon which we are about to operate.

The process terminates when $\boldsymbol{A}_r\boldsymbol{W}_r^T$ is a diagonal matrix $\boldsymbol{D}$, i.e. when the rows of $\boldsymbol{A}_r$ and $\boldsymbol{W}_r$ form a bi-orthogonal set. As $\boldsymbol{W}_r$ is orthogonal, this implies that $\boldsymbol{A}_r = \boldsymbol{DW}_r$, i.e. each row of $\boldsymbol{A}_r$ is a scalar multiple of the corresponding row of $\boldsymbol{W}_r$. This requirement can be used as a check on the accuracy of the computation. The eigenvectors of $\boldsymbol{A}$ are the rows of $\boldsymbol{W}_r$, while the eigenvalues are the elements of $\boldsymbol{D}$. Hence each row of $\boldsymbol{A}_r$ is an eigenvector scaled by its corresponding eigenvalue.

In the conventional Jacobi method each transformation $\boldsymbol{A}_r \rightarrow \boldsymbol{A}_{r+1}$ involves two multiplications for each modified element, hence $4n$ multiplications altogether ($4n$ elements are modified, but they are equal in pairs). If the eigenvectors are to be calculated as well as the eigenvalues, an additional $4n$ multiplications are required to calculate the modified elements of the matrix $\boldsymbol{W}_r$.

In our method each transformation involves two rows, each containing $2n$ elements, hence $8n$ multiplications are required. In addition, we must form three scalar products of rows of n elements, involving another $3n$ multiplications. The number of operations is therefore 37% greater than in the regular method when the eigenvectors are required. In Method I, unlike the standard Jacobi method, no reduction in number of operations is possible when the eigenvectors are not wanted.

Method II

This method is based on the equations

$$\begin{aligned} \boldsymbol{A}_{r+1} &= \boldsymbol{V}_r\boldsymbol{A}_r \\ \boldsymbol{B}_r &= \boldsymbol{A}_r\boldsymbol{A}_r^T \end{aligned} \tag{8}$$

hence

$$\boldsymbol{B}_{r+1} = \boldsymbol{V}_r\boldsymbol{B}_r\boldsymbol{V}_r^T. \tag{9}$$

We compute the sequence $\boldsymbol{A}_r$, $r = 1, 2, \ldots$, with $\boldsymbol{A}_1 = \boldsymbol{A}$, and choose the angles θ_r in such a way that $\boldsymbol{B}_r$ converges to a diagonal matrix. The technique is therefore essentially the diagonalization of the matrix $\boldsymbol{B} = A^2$ by the normal Jacobi method, using the very same rotations that would be used for this purpose by the commonly used method, but carrying out all the calculations on the matrix $\boldsymbol{A}$.

The procedure for performing the rth step is to form from $\boldsymbol{A}_r$ the elements b_{pp}, b_{qq}, and b_{pq} of $\boldsymbol{B}_r$. This involves forming the norm* (sum of squares of elements) of the pth and qth rows of A_r, and the scalar product between the pth and qth rows, a total of $3n$ multiplications. From these numbers we compute the angle θ_r in the normal way, then replace the pth and qth rows of A_r by their appropriate linear combinations according to eqn. (5), so obtaining A_{r+1}; this latter operation involving $4n$ multiplications.

The reduction is terminated when all pairs of rows of A_r are mutually orthogonal, i.e. when A is of the form DW. The orthogonal matrix W that appears here is identical to the product of the matrices V_r that have been used to bring A to this form—this is a consequence of the symmetry of A. It is because it uses this fact, and thereby eliminates the need to compute W separately, that Method II is simpler than Method I. It should be noted that, although the elementary matrices V_r are different, the matrix W is the same in the two methods, for the eigenvectors of A^2 are identical to those of A. We now have the eigenvectors as the (suitably normalized) rows of W, and the eigenvalues as the square roots of the norms of these rows.

We have estimated the number of operations required for each transformation as $7n$, but the $3n$ which are used to calculate the angle θ_r can be reduced to n by the simple expedient of recording the norm of each row as an additional item along with the row. Whenever we operate on two rows with eqn. (5) we then also re-calculate their norms from the formulae†:

$$\left.\begin{aligned} b_{pp}^{(r+1)} &= b_{pp}^{(r)} \cos^2\theta - 2b_{pq}^{(r)} \cos\theta \sin\theta + b_{qq}^{(r)} \sin^2\theta \\ b_{qq}^{(r+1)} &= b_{pp}^{(r)} + b_{qq}^{(r)} - b_{pp}^{(r+1)} \end{aligned}\right\} \quad (10).$$

As the repeated use of this formula can result in a gradual accumulation of error in the row norms, especially in those which are becoming the smaller eigenvalues of the matrix B, and as the rate of convergence can be drastically reduced by errors in the computation of the angles θ_r, it is wise to recompute the norm of each row from the elements of the row after, say, every sweep through the matrix. In any case, it should be possible to carry out this re-calculation sufficiently infrequently for us to estimate the number of operations per step as $5n$ in place of $7n$. This is to be compared with the $8n$ required by the usual Jacobi method when the eigenvectors are required, or $4n$ when they are not.

On the grounds of speed alone Method II is therefore to be preferred to Method I. However, Method II has three weaknesses not shared by Method I. The first is that, in yielding the eigenvalues as scaling factors associated with the eigenvectors, it yields only their magnitudes and not their signs. (In essence, they are obtained as the square roots of the eigenvalues of A^2.) This difficulty can be met by augmenting the matrix A with a column of the unit matrix. In Method I we used the complete unit matrix, obtaining (WA, W) from (A, I), and then determined the eigenvalues from the ratio of corresponding elements in WA and W. Obviously, it is not necessary for this purpose to compute the whole matrix W. If we augment A with the first column alone of the unit matrix we will obtain the first column of W and, except for any rows of W which may have a zero element in this column, can then determine the sign of the eigenvalue associated with each row.

A second weakness of Method II lies in the fact that all eigenvectors are obtained scaled by their corresponding eigenvalues. Consequently the eigenvector associated with a zero eigenvalue is not obtained at all, while eigenvectors associated with very small eigenvalues are obtained very inaccurately. This difficulty, if anticipated, can be avoided by adding a suitable multiple of the unit matrix to A so as to shift all eigenvalues away from the origin.

The third* source of difficulty with Method II is, like the first, also due to the squaring of the eigenvalues. If the matrix A has two eigenvalues of equal magnitude but opposite sign these become, on squaring, a multiple eigenvalue of A^2. The corresponding eigenvectors, as calculated, are then unlikely to be eigenvectors of A, but will lie in the space spanned by the corresponding two eigenvectors of A. This trouble also can be avoided, if foreseen, by a shift of origin.

3. Applications to Non-symmetric and Rectangular Matrices

As neither of our two methods, when considered purely as an algorithm, is explicitly dependent upon the symmetry of the matrix A—in fact they both destroy this symmetry in the first transformation—let us consider what their effect is when applied to non-symmetric matrices. Of course, we will not obtain the eigenvalues of a non-symmetric matrix in this way, but it is of interest to see what we will obtain.

Let us, very briefly, analyse the two methods again in terms of a non-symmetric A.

Method I:

We choose W so that rows of WA are orthogonal to rows of W. When operating on rows p and q we can choose θ_r so that either row p of WA is orthogonal to row q of W, or row q of WA is orthogonal to row p of W. If A is symmetric it makes no difference which choice we make—if one pair is orthogonal the other will be too—but this is not so in the case of a non-symmetric matrix. In terms of the product WAW^T, we make only one of the two off-diagonal elements (p, q) and (q, p) zero. By choosing, say, always that element which is below the diagonal, we are striving to make WAW^T a triangular matrix.

Method I therefore provides us with a technique for triangulating non-symmetric matrices by the methods studied by Greenstadt (1955), Lotkin (1956), Causey

* This is not a true "norm" according to the rigorous definition of the word. The term is used here for convenience, no more suitable word being available.

† Equations (10) are immediate consequences of equations (5) and the definition of b_{pp}, b_{qq}, and b_{pq}.

* I am indebted to the referee for pointing out this fact.

(1958) and Dimsdale (1958), using a serial-access store. Of course, it is also necessary to use unitary transformations in place of the orthogonal ones we have been speaking of. If the difficulties at present being experienced in ensuring the convergence of these methods are overcome, this application of our method may become of importance.

Method II:

As we are essentially diagonalizing the matrix $\boldsymbol{B} = \boldsymbol{AA}^T$ which is always symmetric, the method will converge for non-symmetric matrices and will yield the singular values* of $\boldsymbol{A}$ (i.e. the square roots of the eigenvalues of $\boldsymbol{B}$). We will have performed the transformation

$$\boldsymbol{WA} = \boldsymbol{DX}$$

where now the orthogonal matrix $\boldsymbol{X}$ is not identical to $\boldsymbol{W}$. The rows of $\boldsymbol{W}$ are the eigenvectors of $\boldsymbol{AA}^T$, while the rows of $\boldsymbol{X}$ are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$.

Method II, unlike Method I, can also be applied to a rectangular matrix $\boldsymbol{A}$. As in the case of a non-symmetric square matrix, it yields the eigenvalues and vectors of the symmetric matrices $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{AA}^T$. The technique may therefore have a useful application in factor analysis, operating directly on the scores instead of the correlation matrix, and also in the analysis of under- and over-determined systems of linear equations.

4. A Variation in the Sequencing of Transformations

The sequence in which the off-diagonal elements a_{pq} are selected for elimination in the Jacobi method is normally either that of their magnitude or in the fixed sequence:

$$\begin{aligned}(p, q) = {} & (1, 2), (1, 3), \ldots, (1, n),\\ & (2, 3), (2, 4), \ldots, (2, n),\\ & (3, 4), \ldots\\ & \ldots (n-1, n).\end{aligned}$$

In this "cyclic" (see Henrici, 1958) method every sub-diagonal element is chosen once in an iteration† of $n(n-1)/2$ transformations.

In our modification of the Jacobi method we can use precisely the same sequencing, but now p and q represent the two rows of the matrix upon which we perform the transformation. However, as we shall see in the next section, this particular cyclic arrangement cannot be efficiently programmed for use in computers which have only a small number of tape units, unless the tapes can be read and written while moving in both the forward and backward directions.

An alternative sequencing, which can be more efficiently programmed, consists of choosing the pairs of rows in the order

$$q = p+1, p+2, \ldots, n, 1, 2, \ldots, p-1$$
$$\text{for } p = 1, 2, \ldots, n.$$

i.e.

$$\begin{aligned}(p, q) = {} & (1, 2), (1, 3), \ldots, (1, n-1), (1, n),\\ & (2, 3), (2, 4), \ldots, (2, n), (2, 1),\\ & (3, 4), (3, 5), \ldots, (3, 1), (3, 2),\\ & \quad \cdot \quad \cdot \quad \cdot\\ & (n, 1), (n, 2), \ldots, (n, n-2), (n, n-1).\end{aligned}$$

In one iteration we perform $n(n-1)$ transformations, and every pair of rows is operated upon twice. We shall therefore call this method the *double-iteration method* and the conventional cyclic arrangement the *single-iteration method.*

The double-iteration method may appear at first acquaintance to be somewhat inefficient in that the operations performed upon any given pair of rows do not occur at equally spaced intervals, but it is not so. Consider, for example, the first and second rows. The pair (1, 2) is operated upon in the first and again in the $(2n-2)$th transformation (see first and second rows of the table above). This suggests that the second transformation upon this pair, appearing so soon after the first, is, in a sense, a wasted operation. However, a count of the operations performed upon each row shows that, between the appearance of the pair (1, 2) and the pair (2, 1) in the table, both rows 1 and 2 are orthogonalized once with respect to each of the other $n-2$ rows of the matrix. Similarly, between (2, 1) and (1, 2), although a much larger total number of operations is performed, again both rows 1 and 2 are orthogonalized once with respect to each of the other $n-2$ rows. Hence, although the orthogonalizations of the pair (1, 2) do not occur at equally spaced intervals of time, they do occur at precisely those times at which they will be most effective. It can easily be seen that the very same reasoning applies also to all other pairs of rows.

Techniques for applying both the single and double-iteration schemes are outlined in the following section.

5. Sequencing the Magnetic-tape Operations

Let us first indicate the sequence of operations for carrying out the double-iteration scheme. To provide a specific example we will consider a matrix of order seven. (We choose a small matrix, for which the method would be inefficient, in order to simplify the exposition.) We start with the matrix recorded row by row on a single tape, which we call the *input tape*. We shall

* Householder (1958) uses the term "singular values" for the positive square roots of the eigenvalues of $\boldsymbol{AA}^T$, while Forsythe and Henrici (1960) call them the "principal values." The largest singular value is the "spectral norm."

† We shall use the term *iteration* to indicate a complete cycling through all off-diagonal elements of the matrix. The term *sweep* will mean a process in which the entire matrix is transferred from one magnetic tape to another. One iteration generally consists of n sweeps.

assume that each row forms a single block, or a whole number of blocks, on the tape. This enables us to speak of "reading a row" and "writing a row" as single operations.

The first six row operations involve the row pairs (1, 2), (1,3), (1,4), (1,5), (1,6), (1,7). Hence we read row 1, then read each of the other rows in turn, perform the operation, and write it on the *output tape*. After completing this sweep we write row 1 on the output tape. Thus the first sweep has transferred the matrix from one tape to another with a cyclic permutation of its rows from the sequence 1, 2, 3, 4, 5, 6, 7, to 2, 3, 4, 5, 6, 7, 1.

If we now rewind both tapes and repeat exactly the same procedure, but with the roles of the two tapes reversed, we will perform operations on the row pairs (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 1) and will leave the rows in the sequence 3, 4, 5, 6, 7, 1, 2. Obviously seven such sweeps constitute one complete double-iteration, as defined earlier, and also restore the rows to their original sequence.

If the computer has two tape units we would use them in the manner described above. If the computer has only a single tape unit we can simulate two tapes on the one by using alternate blocks for input and output. We start with the rows of the matrix recorded on alternate blocks as follows:

1 – 2 – 3 – 4 – 5 – 6 – 7

where the hyphens indicate spaces* left for recording the output of the first sweep.

On the first sweep down the tape row 1 is held in the store while the other rows are read, operated upon, then written in the following block. While operating on row 7 we rewind the tape which now has the format

– – – 2 – 3 – 4 – 5 – 6 –

The new row 7 is then recorded in the first block, row 1 in the third, then row 2 is read from the fourth block and held in the store while rows 3, 4, 5 and 6 are read, operated upon, and recorded. The arithmetic unit must now wait while the tape, which has the following format, is being rewound:

7 – 1 – – – 3 – 4 – 5 – 6

Rows 7 and 1 are then each read, operated upon, and recorded in the following block. Then row 2 is recorded in the sixth block, and so on. It can be seen that some computation is done during alternate rewind operations. If the entire process is followed through it will be seen that eight and one half sweeps of the tape are required to perform a complete double-iteration, and that the tape will be left in the form

– 4 – 5 – 6 – 7 1 – 2 – 3

* On most computers it is not possible to leave such gaps on the tape, but the same result can be obtained by recording blocks of the appropriate number of words. If the design of the magnetic-tape system is such that the recording of a block may corrupt the neighbouring block it may be necessary to have short "protector" blocks recorded between those which carry the matrix rows.

with the reading head lying between rows 7 and 1 ready to start another iteration. The number of rewind operations therefore differs by a negligible amount from the number required when two tape units are available, but, of course, each rewind extends over twice the length of tape. We therefore deduce that in a computer that does not allow of autonomous tape operations (apart from rewind) a single tape unit is less efficient than a pair only in that approximately twice as much time is spent in rewind operations.

If the computer has four tape units we can eliminate completely the wastage of time in rewind operations by storing only one half of the rows of the matrix on each tape. Let us refer to the four tapes as *A*, *B*, *C* and *D*, and let us start with rows 1, 2, 3, 4 on tape *A* and 5, 6 and 7 on *B*. In the first sweep we transfer rows 2, 3, 4, 5 to tape *C* and rows 6, 7, 1 to *D*. Then, for example, in the first sweep, as soon as row 4 has been read, rewinding of tape *A* is commenced and the input is switched to *B*. At the end of the first sweep, when tape *B* starts rewinding, tape *A* should be fully rewound and ready to act as an output tape. Similar reasoning applies to tapes *C* and *D*.

The amount of computation involved in each row operation is of the order of five arithmetic operation times* per element, and, for the performance of each row operation, we must read one row from tape and write one on tape. Thus, for each step, we perform on each element of the matrix five arithmetic operations and two tape transfers. If the two tape transfers take very much less time than the five arithmetic operations, then the computation is "computer bound" and any questions of making optimum use of the tapes are irrelevant. In this case either one, two or four tape units used in the manner described above would be approximately equally efficient, and the program will be quite simple. Conversely, if the computation is "tape bound," then our technique is hardly suitable at all unless the computer has a large number of tapes and can operate at least one half of them simultaneously. We shall not discuss the details of such a situation.

The most interesting situation is that in which the arithmetic operations and the tape transfers take approximately equal times. Maximum efficiency can then be achieved only if tape transfers can be carried out simultaneously with arithmetic computation. If one tape transfer takes the same or less time than 2·5 operation times, then the two operations of reading a row and writing a row can be performed consecutively during one row operation. If a single tape transfer takes longer than 2·5 operation times then, for maximum efficiency, we shall want to carry out two tape operations, one writing and the other reading, simultaneously with each other and with an arithmetic computation.

The sequencing of operations when either one or two tape operations can be carried out autonomously is the

* We define an "operation time" as the time for one multiplication plus an addition and the associated "book-keeping" instructions.

Table 1

Sequencing the Double-iteration Algorithm on Four Magnetic-tape Units

COLUMN NO.	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
STEP NUMBER	OPERATE ON ROWS	INPUT ROW	OUTPUT ROW	INPUT TAPE	OUTPUT TAPE	OPERATE ON BLOCKS	INPUT BLOCK	OUTPUT BLOCK
1	—	1	—	*A*	—	—	*a*	—
2	—	2	—	*A*	—	—	*b*	—
3	1, 2	3	—	*A*	—	*a*, *b*	*c*	—
4	1, 3	4	2	*A*	*C*	*b*, *c*	*d*	*a*
5	1, 4	5	3	*B*	*C*	*c*, *d*	*a*	*b*
6	1, 5	6	4	*B*	*C*	*d*, *a*	*b*	*c*
7	1, 6	7	5	*B*	*C*	*a*, *b*	*c*	*d*
8	1, 7	2	6	*C*	*D*	*b*, *c*	*d*	*a*
9	—	3	7	*C*	*D*	—	*a*	*b*
10	2, 3	4	1	*C*	*D*	*d*, *a*	*b*	*c*
11	2, 4	5	3	*C*	*A*	*a*, *b*	*c*	*d*
12	2, 5	6	4	*D*	*A*	*b*, *c*	*d*	*a*
13	2, 6	7	5	*D*	*A*	*c*, *d*	*a*	*b*
14	2, 7	1	6	*D*	*A*	*d*, *a*	*b*	*c*
15	2, 1	3	7	*A*	*B*	*a*, *b*	*c*	*d*
16	—	4	1	*A*	*B*	—	*d*	*a*
17	3, 4	5	2	*A*	*B*	*c*, *d*	*a*	*b*
18	3, 5	6	4	*A*	*C*	*d*, *a*	*b*	*c*
19	3, 6	7	5	*B*	*C*	*a*, *b*	*c*	*d*
20	3, 7	1	6	*B*	*C*	*b*, *c*	*d*	*a*
21	3, 1	2	7	*B*	*C*	*c*, *d*	*a*	*b*
22	3, 2	4	1	*C*	*D*	*d*, *a*	*b*	*c*
23	—	5	2	*C*	*D*	—	*c*	*d*
24	4, 5	6	3	*C*	*D*	*b*, *c*	*d*	*a*
25	4, 6	7	5	*C*	*A*	*c*, *d*	*a*	*b*
26	4, 7	1	6	*D*	*A*	*d*, *a*	*b*	*c*
27	4, 1	2	7	*D*	*A*	*a*, *b*	*c*	*d*

same and is illustrated in columns 1 to 5 of Table 1. As there is some overlap between input and output operations of consecutive sweeps (due to the fact that rows are read in before they are required and written after being operated upon) we find in our example that a tape unit occasionally has to act as an input tape immediately after its use as an output tape, leaving no time for rewind: for example, tape *C* between steps 7 and 8 in Table 1. However, this effect is due to the small size of the matrix we have taken as an example. During the rewinding of an output tape in preparation for its use as an input tape a total of $\frac{1}{2}n-3$ rows are read from (and written on) other tapes. Hence, with large matrices the rewind operation needs to be only a few per cent faster than the reading and writing operations. It can also be seen from Table 1 that there are exactly three steps in each sweep (i.e. three steps out of n, where n is the number of rows in the matrix) in which the two tapes being operated upon are both members of the same pair. Thus, in the first sweep, tapes *C* and *D* are both operated upon in steps 8, 9 and 10. In all other steps of each sweep we use one tape of the pair *A* and *B*, together with one of the pair *C* and *D*. It follows that a computer, in which the tapes are in two groups such that only one tape of each group can be used at one time, is very nearly as efficient for our purposes as one in which any two tapes can be used simultaneously.

In order to be able to carry out an arithmetic operation on two rows, read a row and write a row, all simultaneously, we need to reserve four blocks of storage in the working store, each large enough to hold one row of the matrix. We shall refer to these blocks as *a*, *b*, *c* and *d*. From the point of view of the efficiency of the magnetic-tape operations it is unimportant how we organize the use of the four blocks of data in the main store, but this organization is important from the point of view of the simplicity of the program. The organization of the program is greatly simplified by arranging that, whenever two rows are operated upon, their positions in the store are interchanged. That is, if row 1 is in block *a* and row 2 is in *b*, then after operating upon these two rows we leave the new row 1 in *b* and the new row 2 in a. As the operation consists of replacing each pair of elements by a pair of linear combinations of the same two elements, it is just as easy to interchange the rows in this way as it is to leave them in place.

The last three columns of Table 1 show the way in which the four storage blocks are used. For example, in step 3 we have row 1 in a, row 2 in b and row 3 being read into c. After the operation on rows 1 and 2 has been performed, we will have the new row 2 in a and the new row 1 in b. Hence in step 4 we read out row 2 from a and operate on rows 1 and 3 in b and c.

It can now be seen how very simple the procedure is. The four blocks a, b, c and d are always used in exactly that order as both input and output blocks. Hence the input program must cycle through the four blocks (see column 7 in Table 1), giving a cycle of length four, and also through the four tape units with a cycle of length $2n$ (see column 4). The output program has an identical cyclic structure, but is "out of phase" with the input program by a fixed amount. The arithmetic program is also based upon a double cycle. It cycles through the four pairs (a, b), (b, c), (c, d), and (d, a) with a cycle of length four, but also counts cycles of length n so as to inhibit every nth operation.* Note that an inhibited operation step is counted as a normal step in determining the cycling through the blocks.

There is no break whatsoever in any of these cycles as we finish one iteration and start another; the only complications are in starting and stopping the entire process. The first few steps differ from the others in that we inhibit the arithmetic procedure in step 1 (it is automatically inhibited in step 2), and the output procedure in steps 1 to 3. However, if we start with tapes A, C and D rewound and tape B located at the end of the file (i.e. at the end of the last row of the matrix), we can then allow the first few steps to be the same as all the others.

To start, therefore, we rewind tapes C and D, read in the matrix from the primary input medium (cards, paper tape, etc.) copying the first half on to tape A, then rewinding tape A while copying the second half on to tape B, then start immediately on the first step of our procedure by setting up the appropriate "phase" in each of the programmed cycles.

We must now consider the problem of obtaining access to the complete matrix after the computation has terminated. The most efficient way of testing for convergence is to keep a running count of the number of consecutive operations which were unnecessary because the rows were already orthogonal. This count is set to zero whenever a non-zero angle θ_r is used, and is increased by unity otherwise. Then when the count reaches $n(n - 1)$ we know that all pairs of rows are orthogonal, hence the procedure has terminated. The program may therefore stop at any point within an iteration. Then if either A or B was the tape last used as an output tape, we will have on tapes C and D a complete copy of the matrix with the rows permuted, but still in cyclic order. Similarly, if either C or D was last used as output we obtain a complete copy from A and B.

The ordering of the rows will be important if an attempt has been made to choose the angles of rotation so that the eigenvalues are ordered according to magnitude.* In such a case the ordering can be restored if we keep a record of the total number of sweeps that have been completed. Each sweep cyclically permutes the rows by one place.

* This cycling of three different programs through a group of four blocks in the store is a procedure which appears to be very amenable to the use of an automatic indexing facility of the nature of the "control words" used in the IBM Stretch (see Blaauw, 1959) and the "record definition words" of the IBM 7070 computer.

6. Sequencing Tape Operations for the Single-Iteration Scheme

The single-iteration scheme operates on row pairs in the sequence

$$(1, 2), (1, 3), (1, 4), \ldots, (1, n)$$
$$(2, 3), (2, 4), \ldots$$
$$\ldots (n - 1, n)$$

where each line represents a sweep. One iteration is therefore made up of n sweeps, as in the double-iteration method, but successive sweeps involve a decreasing number of rows. After the rth sweep rows 1 to r play no further role and only the remaining n–r rows remain active. For efficient use of magnetic tape it is therefore necessary to arrange that the rows are successively placed on an "inactive file" which plays no further role in the current iteration. To simplify starting the next iteration this inactive file should have the same format as the original matrix.

In a multi-tape computer the inactive file can be a separate tape and we can therefore still devise an efficient program. The single-iteration method, however, appears to be rather inefficient on a one-tape computer unless the machine has the facility of being able to read and write while the tape is moving in either direction. The following method was devised for the SILLIAC, which has a single tape unit which allows of reading and writing in both directions, but does not allow of simultaneous tape transfers and arithmetic operations.

We start with the rows of the matrix recorded on alternate blocks on the tape, thus:

$$1 - 2 - 3 - 4 - 5 - 6 - 7 -$$

(as before, we are using a seven row matrix as an example), and we reserve three blocks of storage in working store, called a, b and c. To begin the first sweep we read row 1 into a, row 2 into b, then operate on these two rows. Rows 3, 4, 5, 6 and 7 are now read, in turn, into c, operated upon with row 1 (which is in a),

* A technique for obtaining the eigenvalues in an ordered sequence is described in: B. A. Chartres, "Computing Extreme Eigenvalues of Real Symmetric Matrices." Technical Report No. 9, Basser Computing Department, University of Sydney.

then written back on to the tape in the succeeding empty block position.

At the end of the first sweep we still have row 1 in *a*, 2 in *b*, and the tape is as follows:

$$- - - - - 3 - 4 - 5 - 6 - 7$$

In the second sweep, which is performed with the tape moving in the reverse direction, we read each of the rows 7, 6, 5, 4, and 3, in turn, into *c*, operate upon it with row 2 (which is in *b*) and write it back on to the tape in the position it originally occupied. Then we write row 2 and row 1 back into their original positions. The configuration of the tape is now exactly the same as it was originally, and we have completed two sweeps. The tape is now moved forward again to the beginning of row 3, the count number kept by the program to indicate the number of rows which are active is reduced by two, and exactly the same procedure is repeated over again.

When the number of active rows has been reduced to two or three (depending upon whether the original number of rows was even or odd) the procedure is changed, for now the calculation can be done entirely within the working store. At the end of the iteration the tape is rewound and a second interation started. It should be noted that the sequence in which we have chosen pairs of rows is not quite the same as the usual single-iteration scheme, being:

$$\begin{gathered}(1, 2), (1, 3), (1, 4), \ldots, (1, n), \\ (2, n), (2, n-1), \ldots, (2, 3), \\ (3, 4), (3, 5), \ldots, (3, n), \\ (4, n), \ldots, \text{etc.}\end{gathered}$$

The difference should have no effect on the rate of convergence.

A count of the number of operations involved shows that there are $n(n-1)/2$ row operations performed per iteration, each operation involving one read and one write operation. In addition there are $n/2$ additional read operations, $n/2$ write operations, $4n$ blocks on the tape are moved over in wasted motion, and only one rewind is performed. The method is therefore considerably more efficient than the methods described earlier for the double-iteration procedure on one-tape and two-tape machines. The increased efficiency is, of course, due to our using the ability of SILLIAC to read and write on tapes in both the backward and forward direction, thereby eliminating unnecessary rewind operations.

An adaptation of this method to a computer in which the tape transfers are autonomous and bi-directional is

Table 2

Sequencing the Single-iteration Algorithm on Six Magnetic-tape Units

COLUMN NO.	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
STEP NUMBER	OPERATE ON ROWS	INPUT ROW	OUTPUT ROW	INPUT TAPE	OUTPUT TAPE	OPERATE ON BLOCKS	INPUT BLOCK	OUTPUT BLOCK
1	—	1	—	*E*	—	—	*a*	—
2	—	2	—	*E*	—	—	*b*	—
3	1, 2	3	—	*E*	—	*a*, *b*	*c*	—
4	1, 3	4	2	*E*	*A*	*b*, *c*	*d*	*a*
5	1, 4	5	3	*F*	*A*	*c*, *d*	*a*	*b*
6	1, 5	6	4	*F*	*A*	*d*, *a*	*b*	*c*
7	1, 6	7	5	*F*	*B*	*a*, *b*	*c*	*d*
8	1, 7	2	6	*A*	*B*	*b*, *c*	*d*	*a*
9	—	3	7	*A*	*B*	—	*a*	*b*
10	2, 3	4	1	*A*	*E*	*d*, *a*	*b*	*c*
11	2, 4	5	3	*B*	*C*	*a*, *b*	*c*	*d*
12	2, 5	6	4	*B*	*C*	*b*, *c*	*d*	*a*
13	2, 6	7	5	*B*	*C*	*c*, *d*	*a*	*b*
14	2, 7	3	6	*C*	*D*	*d*, *a*	*b*	*c*
15	—	4	7	*C*	*D*	—	*c*	*d*
16	3, 4	5	2	*C*	*E*	*b*, *c*	*d*	*a*
17	3, 5	6	4	*D*	*A*	*c*, *d*	*a*	*b*
18	3, 6	7	5	*D*	*A*	*d*, *a*	*b*	*c*
19	3, 7	4	6	*A*	*B*	*a*, *b*	*c*	*d*
20	—	5	7	*A*	*B*	—	*d*	*a*
21	4, 5	6	3	*B*	*E*	*c*, *d*	*a*	*b*

easily deduced. The arithmetic unit can be kept continuously engaged if the tape transfer rate is sufficiently fast to enable one block to be read, one written, and one passed over during one row operation. That is, the tape transfer rate should be such that three words can be transferred during five arithmetic operation times. A full description of this adaptation does not appear to be warranted.

For computers able to read and write in only the forward direction, the methods described earlier for the double-iteration system on two-tape and four-tape machines can be adapted for the single-iteration scheme by using an additional tape unit for the "inactive file." Table 2 shows the scheme of Table 1 adapted for the single-iteration procedure on a six-tape computer. The six tapes have been labelled A, B, C, D, E and F, with tapes E and F being used for the inactive file, and therefore also for the initial input. If only one tape had been used for the inactive file there would be a delay while the tape was rewinding between steps 7 and 10, and a similar delay at the end of the iteration. As this delay occurs only twice in each iteration, a five-tape machine would be very little less efficient than the six-tape computer.

The scheme of Table 2 can be continued until there are only four active rows left. Then the remainder of the iteration should be carried out entirely within the working store. The very same simple cycling of the input, output and arithmetic programs among the four storage blocks can be seen in Table 2 as in Table 1.

To summarize the applicability of the schemes we have described two different computer configurations:

(1) For a computer which can read and write on its tapes in both directions, a highly efficient procedure using the single-iteration scheme (slightly modified) exists which employs only one tape unit. This procedure is ideal even for a multi-tape computer with autonomous tape operations.

For a computer in which the tapes will operate only in the forward direction the following remarks apply, irrespective of whether the tape operations are autonomous or not.

(2) For a one-tape or two-tape computer the double-iteration scheme is applicable. A rewind over $2n$ rows in the first case and n in the second case is required every sweep.

(3) For three-tape computers we can use the single-iteration scheme, using one tape for the inactive file, the other two as in (2) above. The amount of time wastage in rewinds is the same as in (2).

(4) For four-tape computers the procedure of Table 1, based on the double-iteration scheme, involves no time wastage in rewind. This method would be preferable to method (3), on such a computer.

(5) For five-tape and six-tape computers the single-iteration scheme of Table 2 can be used, but it does not appear to have any advantage over method (4), which would use only four of the tapes.

7. Summary of Conclusions

We have proposed two methods of carrying out the Jacobi diagonalization procedure for symmetric real matrices. The first method is equivalent to the normal Jacobi process, differing from it only in that the transformed matrix is held in a factored form. The second method, which requires less than one half as many arithmetic operations, is equivalent to the application of the standard Jacobi process to the square of the matrix.

The methods are specially adapted for the use of a magnetic-tape backing store in conjunction with an immediate-access store which need be only large enough to hold a few rows of the matrix. Procedures have been described for a computer with from one to six tape units. A computer which is able to read and write on its tapes with the tape moving in either the forward or backward direction, can be programmed to make very efficient use of a single tape unit. For computers with mono-directional tapes, six tape units are required for optimum efficiency.

Owing to the necessity of organizing access to the matrix rows in a regular manner, the selection of off-diagonal elements for elimination must be performed in a certain pre-determined sequence. The number of tape units required for most efficient operation can be reduced from six to four by using a sequence which differs from the one customarily employed. For the same reason, by-passing the elimination of elements which are already much smaller than the average will not save any time, though it should still be done in order to minimize the rate of accumulation of rounding errors.

If the transfer of data between the main store and the tapes can be performed simultaneously with the carrying out of independent arithmetic operations, then the computer can be kept continuously supplied with the data it needs if the effective data transfer rate is as fast as one word transferred to every 2·5 arithmetic operation times. (An "arithmetic operation time" is the time for one multiplication, plus one addition, plus the associated book-keeping instructions.) This transfer rate can be achieved either through a single channel operating at the stated speed, or through two channels, one reading from and the other writing on tape, each operating at one half that speed. Our second method, which is the faster one, will then be somewhat faster than the usual Jacobi method.

The second method, when used for the calculation of both eigenvalues and eigenvectors, is both faster and uses less total storage space than the regular Jacobi procedure, and may therefore be preferable to it even when the entire calculation is performed within the main working store.

This work was performed in part at the School of Physics, University of Sydney, and in part in the Division of Applied Mathematics, Brown University, where it was supported by a grant from the International Business Machines Corporation.

References

Blaauw, G. A. (1959). "Indexing and Control-Word Techniques," *IBM J. Research and Development*, Vol. 3, pp. 288–301.
Causey, R. L. (1958). "Computing Eigenvalues of Non-Hermitian Matrices by Methods of Jacobi Type," *J. Soc. Industr. Appl. Math.*, Vol. 6, pp. 172–81.
Dimsdale, B. (1958). "The Non-Convergence of a Characteristic Root Method," *J. Soc. Indust. Appl. Math.*, Vol. 6, pp. 23–5.
Forsythe, G. E., and Henrici, P. (1960). "The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix," *Trans. Amer. Math. Soc.*, Vol. 94, pp. 1–23.
Givens, W. (1957). "The Characteristic Value-Vector Problem," *J. Assoc. Comput. Mach.*, Vol. 4, pp. 298–307.
Goldstine, H. H., Murray, F. J., and von Neumann, J. (1959). "The Jacobi Method for Real Symmetric Matrices," *J. Assoc. Comput. Mach.*, Vol. 6, pp. 59–96.
Greenstadt, J. (1955). "A Method for finding Roots of Arbitrary Matrices," *Math. Tab. Wash.*, Vol. 9, pp. 47–52.
Gregory, R. T. (1953). "Computing Eigenvalues and Eigenvectors of a Symmetric Matrix on the ILLIAC," *Math. Tab. Wash.*, Vol. 7, pp. 215–20.
Henrici, P. (1958). "On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing Eigenvalues of Hermitian Matrices," *J. Soc. Indust. Appl. Math.*, Vol. 6, pp. 144–62.
Householder, A. S. (1953). *Principles of Numerical Analysis*. New York: McGraw-Hill Book Co.
Householder, A. S. (1958). "The Approximate Solution of Matrix Problems," *J. Assoc. Comput. Mach.*, Vol. 5, pp. 205–43.
Johansen, D. E. (1961). "A Modified Givens Method for the Eigenvalue Evaluation of Large Matrices," *J. Assoc. Comput. Mach.*, Vol. 8, pp. 331–5.
Lotkin, M. (1956). "Characteristic Values of Arbitrary Matrices," *Quart. J. Appl. Math.*, Vol. 14, pp. 267–75.
Pope, D. A., and Tompkins, C. B. (1957). "Maximizing Functions of Rotations," *J. Assoc. Comput. Mach.*, Vol. 4, pp. 459–66.
Rollett, J. S., and Wilkinson, J. H. (1961). "An Efficient Scheme for the Co-diagonalization of a Symmetric Matrix by Givens' Method in a Computer with a Two-level Store," *The Computer Journal*, Vol. 4, pp. 177–80.
Wilkinson, J. H. (1958). "On the Calculation of Eigenvectors of Co-diagonal Matrices," *The Computer Journal*, Vol. 1, pp. 90–6.

Book Review

Theory of the Transmission and Processing of Information, by A. G. Vitushkin. Translated from the Russian by Ruth Feinstein, 1961, 206 pages. (London: *Pergamon Press Limited*, 100s.)

This is a fragment of the mathematical theory of approximation, following work of A. N. Kolmogorov and others.

The title "Theory of Transmission and Processing of Information" is misleading, being copied from the first sentence of the Foreword. This states that the monograph is connected with that theory; but the connection seems rather weak. A better title appears at the head of alternate pages of the text, viz. "Complexity of Tabulation Problems." The author asserts that the formal definition of the concept of the complexity of a tabulation problem (i.e. the construction of tables for functions) is required in the automatization of programming, but does not explain why.

The monograph is concerned with estimating the "complexity" of tables of certain general classes of functions. A "table" of a function $f(x)$, defined over a range G of values of x, is understood to mean an ordered set of quantities $y_1, y_2, \ldots, y_p$, each represented to a finite accuracy of (say) n binary digits, together with a "decoding rule" whereby any value of x in the range G is used to enter the table and yield a value $\phi(x)$. This is to differ from $f(x)$ by at most ϵ, the "accuracy" of the table, for each x in G. This decoding rule is in general to be a polynomial in the variables $y_1, \ldots, y_p$, of degree at most k in each one, but its coefficients may depend in any way upon x. As a very simple example, G might be the range $0 \leqslant x \leqslant p$ and $f(x)$ might differ from the constant y_i by less than ϵ in the range $i - 1 \leqslant x \leqslant i$, for $i = 1, 2, \ldots, p$. Then the polynomial could be $\sum_{i=1}^{p} u_i(x).y_i$, where $u_i(x) = 1$ for $i - 1 \leqslant x \leqslant i$ and $u_i(x) = 0$ elsewhere, so that $k = 1$ in this case. The numbers p and k are taken as measuring the "complexity" of the table, and the total number of binary digits employed in it, i.e. np, is called the "volume" of the table. Any set of values $y_1, \ldots, y_p$, each of n binary digits will define some function $\phi(x)$ when used with a fixed decoding rule, and n^p different such functions are possible, forming a "function space" Φ. If a set F of functions $f(x)$ and a number ϵ is given, there will be a certain smallest number N of functions $\phi(x)$ which form an "ϵ-net" of F, i.e. which contain for any f in F, a representative ϕ serving to approximate within ϵ to f throughout G. Then $\log_2 N$ is called the "ϵ-entropy" of the set F with respect to Φ, and it is shown that this is (within 1 unit) the volume of the smallest possible table for functions f in F.

The ϵ-entropy depends on the properties assumed for the functions $f(x)$ which comprise F, as well as on the set Φ of approximating functions. An "absolute" ϵ-entropy is defined which is the lower bound of the above entropy with respect to all possible sets Φ. Estimates are obtained for the absolute ϵ-entropy for some general subspaces of analytic differentiable functions of one or several variables.

These results appear to be of limited interest to users of digital computers.

M. Woodger.