Solution of Systems of Ordinary and Partial Differential Equations by Quasi-Diagonal Matrices

By M. A. Cayless

Methods of solving some rather complicated systems of ordinary and partial differential equations are described, including cases when some of them have eigenvalues or are non-linear. The equations are expressed in finite-difference form as "quasi-diagonal" matrix equations, and subroutines for solving these on the Ferranti Pegasus computer are described, including cases involving latent roots. The examples given are from the theory of gas discharges, but the methods used are of much wider utility.

1. Introduction

In the theory of gas discharges, the need arises for solving systems of simultaneous ordinary or partial differential equations with closed boundary conditions, some of which have eigenvalues. The coefficients of these equations are often quite complicated functions, and often some of the equations are non-linear. Some examples will be found in Cayless (1960) and in Section 5 of the present paper.

We have solved a variety of systems of this type by finite differences, expressing the differential equations in terms of "quasi-diagonal" matrix equations, and have written subroutines for the Ferranti Pegasus computer for solving these, including cases where there are latent roots. These are primarily intended for incorporation in long special-purpose programs for evaluating the characteristics of gas discharges. For this reason we have used the most compact methods we could devise, even at the expense of operating speed, to leave the maximum space for the main program and the large amount of data usually involved.

Although devised primarily for this application, these subroutines are of a quite general nature, and can be used to obtain solutions to a wide variety of equations expressible in this manner.

2. Quasi-Diagonal Matrices

The finite-difference approximation to many systems of linear ordinary differential equations with closed boundary conditions can be arranged in the form of a square matrix in which the only non-zero elements are along the principal diagonal, or its neighbouring "side-diagonals" (see, for example, N.P.L., 1957, or Fox, 1957). In the case of a single second-order equation with the lowest-order difference approximation, there will be one non-zero side diagonal on each side of the principal diagonal thus:

$$\boldsymbol{A}\boldsymbol{y} = \boldsymbol{b} \tag{1}$$

where $\boldsymbol{A}$ has the form

$$\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & a_{32} & a_{33} & a_{34} & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & & \cdot \end{bmatrix}, \tag{2}$$

the remaining elements being zero. For a fourth-order equation, there will be two side diagonals on each side of the principal diagonal, and so on. Such matrices may be termed *quasi-diagonal* matrices, and they enable equations such as (1) to be handled very compactly: in a computer, only the non-zero elements need be stored, together with the elements of the vector $\boldsymbol{b}$.

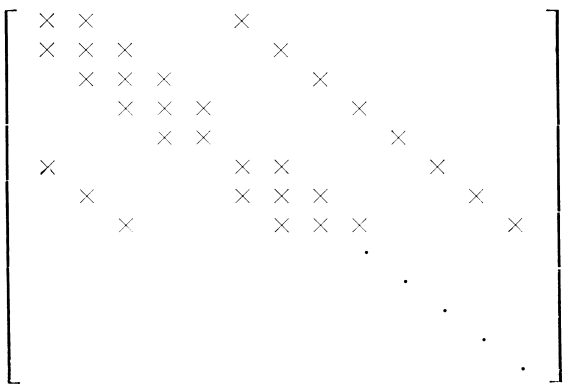
A system of simultaneous differential equations can similarly be arranged in terms of a quasi-diagonal matrix by suitably ordering the variables and equations. For example, two simultaneous second-order equations with dependent variables y and z yield the following form

$$\begin{bmatrix} \times & \times & \times & & & & \\ \times & \times & \times & \times & & & \\ \times & \times & \times & \times & \times & & \\ & \times & \times & \times & \times & \times & \\ & & \times & \times & \times & \times & \times \\ & & & & & \cdot & \\ & & & & & & \cdot \\ & & & & & & & \cdot \end{bmatrix} \begin{bmatrix} y_1 \\ z_1 \\ y_2 \\ z_2 \\ y_3 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} c_1 \\ d_1 \\ c_2 \\ d_2 \\ c_3 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \tag{3}$$

(compare this with Fox, 1957, p. 61: the form given there is not quasi-diagonal).

The finite-difference approximations to partial differential equations can be easily arranged in terms of quasi-diagonal matrices by taking the grid points in a systematic manner. For the two-dimensional second-

order equation corresponding to (1), within a rectangular boundary the matrix has the form



(4)

The number of side-diagonals is equal to the number of grid points in a row within the boundary, and there are correspondingly more in the cases of higher-degree equations and higher-order differences. If the boundary is irregular, the outermost non-zero elements will move from diagonal to diagonal, but the matrix can still be written in quasi-diagonal form, with the furthest non-zero side diagonal in a position corresponding to the longest row of grid points, taking advantage of any symmetry, of course.

Boundary conditions of the general form $py + qy' = r$ are readily included, the only type presenting any difficulty being in the case of curved boundaries when $q \neq 0$: Appendix 1 shows that these also can be arranged to yield quasi-diagonal matrices. Periodic boundary conditions cannot generally be arranged in this way: for an example, see Fox, 1957, p. 93.

There is no restriction to Cartesian coordinates, and curvilinear systems are frequently more appropriate. If the coefficients are at all complicated, the computer can be readily programmed to construct its own matrix, as in Example 2 of Section 5. Equations involving eigenvalues are considered in Section 4.

Since only the quasi-diagonal elements are stored in the computer, very much larger systems can be handled than with a general-purpose scheme for solving any set of simultaneous equations. The solutions are also obtained very much more quickly.

3. Method of Solution

The subroutines we have written for solving equations of this form use direct Gaussian elimination. The elements of the non-zero diagonals [including intermediate zero diagonals, such as those in (4)] of the matrix are stored by rows, together with the right-hand sides (provision is made for a number of vectors, since this is easily arranged). Pivotal condensation is then used, about each diagonal element in turn, to reduce the elements below it to zero, starting with a_{11}, until the matrix is reduced to upper triangular quasi-diagonal form, with unit diagonal elements. Back-substitution then proceeds by the same process, working from the last diagonal element, until the matrix is reduced to the unit matrix. The solution now occupies the locations which originally contained the right-hand side vectors; no drum storage space is required other than that occupied by the original elements of the matrix and vectors.

There are various versions of this procedure in use [see, for example, Livesley (1960), and *Deuce News* (1959), p. 25], and variations are given by Karlquist (1952), Cornock (1954), and Wilson (1959). In its present form the method would become inaccurate if any of the pivotal elements became very small, and fail altogether if one became zero. This eventuality is usually catered for by using row interchanges (see Wilkinson, 1954), but this requires additional storage space and working time in the case of quasi-diagonal matrices. Fortunately it is unnecessary with matrices which are dominated by their diagonals, or which are positive definite, and the matrices arising in practice from differential equations are commonly of this form. In fact, during the present work, in which some hundreds of equations have been solved, no case has been found in which trouble has arisen in this way [compare with the experience of Livesley (1960), p. 38].

The subroutines are very compact; the Pegasus form, working in floating point throughout, occupies 10 Pegasus main store blocks (160 orders) plus the usual floating-point routine (3 blocks); since the latter is invariably needed elsewhere in the computation, we have preferred to make use of it (program space being very limited), rather than include a specially adapted floating-point routine, which would be rather faster.

For the same reason, we have preferred to use the program in its present simple form, rather than to use any more elaborate version, or one of the iterative methods which can deal with very large matrices at very high speeds [see, for example, *Deuce News*, No. 45 (1959), p. 3], but which we have not been able to compress into such a small program space in any general form. (See Appendix 3, however, for a suggested improvement.)

For a matrix of n equations, r side diagonals on each side, and s right-hand side vectors, most of the time is used in evaluating about $rn(r + 2s + 2)$ operations of the form $a'_{ij} = a_{ij} - a_{i,\,i-h}\, a_{i-h,\,j}$. On Pegasus this involves a floating-point multiplication and subtraction, and 4 single word drum transfers. These take an average of about 14, 18 and 4×8 msec respectively, a total of 64 msec per operation. $n(r + s)$ division operations are needed in reducing the pivotal elements to unity, each involving 2 drum transfers and averaging about $18 + 2 \times 8 = 34$ msec. Allowing another few milliseconds for the remaining operations, this gives a total time of approximately $(n/30)[r(2r + 4s + 5) + s]$ seconds, which agrees reasonably well with the times taken in practice.

In the case of partial differential equations it is obviously advantageous to take the grid points in the order that gives the minimum number of side diagonals.

A very much faster version has been written for the Ferranti Mercury computer, in which operations are carried out one whole row at a time in the fast store.

In the case of ordinary differential equations, the solutions can be improved by using the method of the "difference correction"; this is calculated using the same quasi-diagonal matrix equations with new right-hand sides [Fox (1957), p. 35]. A similar method might be possible for partial differential equations, but we have not tried this.

4. Eigenvalue Equations

If we have a differential equation (ordinary or partial) which, together with its boundary conditions, is homogeneous, then it will possess eigenvalues, and the finite-difference equations will be singular. In many cases the finite difference equations have the form

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \tag{5}$$

where $\boldsymbol{A}$ is a quasi-diagonal matrix and λ a constant. If $\boldsymbol{A}$ is of order n there will, in general, be n solutions $\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots$ corresponding to the n latent roots $\lambda_1, \lambda_2 \ldots$ of the determinantal equation.

$$|\boldsymbol{A} - \lambda\boldsymbol{I}| = 0, \tag{6}$$

where $\boldsymbol{I}$ is the unit matrix (NPL, 1957).

The finite-difference approximation is usually a good one for those solutions corresponding to the smaller roots, but gets poorer as the largest root is approached. Fortunately, in many physical problems, it is usually the roots of smaller modulus which are of interest, particularly that of smallest modulus, the "gravest" root: in many cases, including those cited below, only this last has any physical significance.

There are many methods known for extracting latent roots and vectors [see, for example, NPL (1957), and Wilkinson (1958) and (1960)]. The iterative methods described by Fox (1957), pp. 167–9, are well-suited for use with quasi-diagonal matrices, and we have found the following procedure particularly compact and satisfactory for finding the smaller roots and their vectors. We consider only cases where the roots are real and distinct; this is fortunately so in many practical applications arising from differential equations.

If a latent vector $\boldsymbol{x}$ is known approximately, then an approximation to the corresponding root is determined by

$$\lambda_r = \frac{\boldsymbol{x} \cdot \boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x} \cdot \boldsymbol{x}} = \frac{\sum_{ij} a_{ij}x_i x_j}{\sum_i x_i^2}. \tag{7}$$

This method has the advantage that if the matrix is symmetric, then, according to Rayleigh's principle, the value of λ_r so determined is a much better approximation to its true value than is $\boldsymbol{x}$.

To find the vector corresponding to the smallest root, we first note that any arbitrary vector $\boldsymbol{y}$ can be expanded in terms of the complete set of latent vectors $\boldsymbol{x}_i$, thus

$$\boldsymbol{y} = \sum_i \alpha_i \boldsymbol{x}_i, \tag{8}$$

where the α_i are constants and the $\boldsymbol{x}_i$ satisfy

$$\boldsymbol{A}\boldsymbol{x}_i = \lambda_i \boldsymbol{x}_i, \tag{9}$$

where the λ_i are the corresponding latent roots. Thus

$$\boldsymbol{A}^{-r}\boldsymbol{y} = \sum_i \alpha_i \boldsymbol{A}^{-r}\boldsymbol{x}_i = \sum_i \lambda_i{}^{-r}\alpha_i \boldsymbol{x}_i, \tag{10}$$

where r is a positive integer.
Hence the iteration

$$\boldsymbol{y}_{n+1} = \boldsymbol{A}^{-1}\boldsymbol{y}_n \tag{11}$$

will converge, to within an arbitrary factor, to the latent vector $\boldsymbol{x}_1$ corresponding to the smallest latent root λ_1 (provided $\boldsymbol{y}$ is such that $\alpha_1 \neq 0$: this is easily arranged).

Thus $\boldsymbol{x}_1$ can be found by repeatedly dividing the quasi-diagonal matrix $\boldsymbol{A}$ into almost any arbitrary vector $\boldsymbol{y}$, using the program described in Section 3. Once a reasonably good approximation has been obtained, the following procedure provides a highly effective accelerating device.

Supposing a constant p is subtracted from each diagonal of $\boldsymbol{A}$, then from (9)

$$(\boldsymbol{A} - p\boldsymbol{I})\boldsymbol{x}_i = (\lambda_i - p)\boldsymbol{x}_i, \tag{12}$$

and corresponding to (10) we obtain

$$(\boldsymbol{A} - p\boldsymbol{I})^{-r}\boldsymbol{y} = \sum (\lambda_i - p)^{-r}\alpha_i \boldsymbol{x}_i. \tag{13}$$

Clearly, if p is a good enough approximation to any root λ_j, then the iteration

$$\boldsymbol{y}_{n+1} = (\boldsymbol{A} - p\boldsymbol{I})^{-1}\boldsymbol{y}_n \tag{14}$$

will converge to $\boldsymbol{x}_j$ (again, provided $\alpha_j \neq 0$). Once a reasonable approximation is known for $\boldsymbol{x}_j$, we use thereafter the value λ_{jr} obtained from (7) for p; recalculating λ_{jr} after each iteration (14) very rapid convergence to the root is obtained. The convergence of these procedures is discussed by Ostrowski (1958–9).

To obtain the smallest root, if nothing is known, one or two straightforward iterations (11) are used, starting from the 1-vector. λ_r is then calculated by (7) and the above procedure followed. If a good guess can be made of the vector the initial stages may be omitted. Alternatively, if a good guess can be made of the value of the root, the procedure may be started from this, using any suitable starting vector.

Roots higher than the smallest can be obtained by using the iteration (14) directly if a good enough guess can be made to either the vector or the root. A bad guess may lead to the process converging to a neighbouring root; this is readily detected by the form of the vector in many physical problems. We have found in practice that it is very easy to avoid this kind of occurrence, and that the first few roots can readily be determined in this way when the general behaviour of the roots and vectors are known from the physical nature of the problem. Usually from 3 to 6 iterations have been found sufficient to obtain convergence to 6

or 7 significant figures. Occasionally a very large matrix has required 7 or 8 iterations.

Some caution is necessary in using the acceleration procedure since it would fail if the latent root should happen to be exact. This is easily avoided by using a slightly different value for p if (7) should give too accurate an approximation to the exact root, and provision is made in the routine, for arranging this. We have only had to do this once, however, with matrices of any size.

The whole process would fail if the starting vector used were defective in the latent vector sought (i.e. if the appropriate α_i were zero, or very small), and another vector would have to be chosen. This is conceivable, but very unlikely, and easily avoided; it has never happened in our experience.

The Pegasus subroutine for carrying out these procedures operates in floating point throughout, and occupies 21 Pegasus blocks of sixteen orders, in addition to the quasi-diagonal "division" subroutine described in Section 3, and the standard print subroutine. Since these are usually needed elsewhere in the program, the subroutine is very compact. For flexibility, the subroutine is arranged so that the various procedures outlined above can be selected at will by handswitch settings, but, in use, automatic operation can be arranged by including suitable convergence limits. The current approximations to the root and (if required) the vector are printed after each iteration, the latter being normalized so that its maximum element is unity at each stage. To use the subroutine it is simply necessary to specify the size and location of the matrix and vector and (if needed) a starting approximation to the latent root. If no starting vector is specified, a unit vector is assumed.

5. Examples of Solutions

These programs were written principally for solving equations arising in the theory of gas discharges, and the following examples are given to illustrate how they have been used in this field. We have made no attempt to justify the existence and uniqueness of solutions on mathematical grounds, but have made sure that solutions must exist on physical grounds provided the equations are correctly formulated. This is no guarantee that the correct solution will be obtained (the method might fail for one of the reasons mentioned in Sections 3 and 4, for example), but in practice this seems rather unusual, and in the several hundred solutions that have been obtained by these methods in the last two years, we have not had a single case in which the required solution has not been reached when the problem was properly specified. For the derivation of these equations, together with further examples of solutions obtained, see Cayless (1960).

Example 1

In certain gas discharges in mixtures of metal vapours and rare gases, in tubes of any given cross-sectional shape, the equations governing the density of ions and excited atoms are either

$$\nabla^2 x + \lambda x = 0 \tag{15}$$

or

$$\left.\begin{aligned} \nabla^2 x + \lambda x y &= 0 \\ \nabla^2 y + x &= 0 \end{aligned}\right\}, \tag{16}$$

with $x = 0$, $y = 0$ on the wall of the tube. In these x is proportional to the ion density and is to be determined to within an arbitrary factor, and y is proportional to the density of excited atoms. The eigenvalue λ determines the mean electron energy. Equation (15) applies when ionization occurs by the collision of electrons with unexcited atoms, and equation (16) when by collision with the excited atoms whose density is determined by y. Since x is a density, it is essentially positive everywhere, and this leads to the smallest eigenvalue only having any physical significance.

An example of solutions of these equations for the boundary shape shown is illustrated in Fig. 1, x being normalized to have a maximum value of 100. The equations (16) were solved by a program which used the latent root subroutine. The input matrix was assembled by hand and the vector y was put equal to the unit vector. The program divided the matrix row by row by the elements of y, and then evaluated the

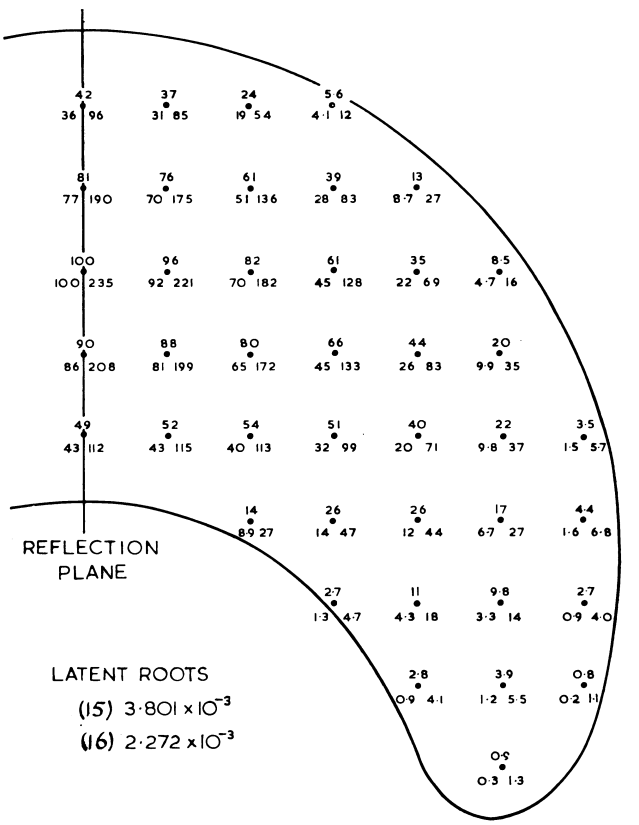


Fig. 1.—Solution of equations (15) and (16) within the boundary shown. Upper figures: values of x for equation (15). Lower figures: values of x and y for equation (16). Both solutions normalized to $x_{max} = 100$

smallest latent root and vector of the resulting matrix. A new vector y was then obtained by dividing the vector x by the original matrix in accordance with the second equation. The procedure was then repeated until it converged. Four iterations were used to obtain the first latent vector x, the acceleration procedure being applied after the first. Subsequently only one iteration of the latent root procedure, with the acceleration procedure, was needed between each recalculation of y. Convergence was obtained to 5 significant figures on the 6th recalculation of y, giving a total number of 15 matrix divisions, taking about 3 minutes each. The whole computation, including monitoring, input, and output, took under an hour. The solution of (15), which could have been obtained by a straightforward application of the latent root routine is, in fact, given by the first stage of the above calculation.

These equations have been solved for a large number of different boundary shapes by this program. In those cases where (15) possessed an analytical solution, the numerical solutions agreed to within 1% with a grid of about 6×6 points over the sector of symmetry.

Example 2

In this example the equations are 5 simultaneous ordinary differential equations in polar coordinates with rather complicated coefficients, the first of which has eigenvalues. They arise in a rather more detailed study of gas discharges in tubes of circular cross-section and are as follows:

$$D_a\left(\frac{d^2n_i}{d\rho^2} + \frac{1}{\rho}\frac{dn_i}{d\rho}\right) + \sum_y z_{yi}n_y n_i = 0, \ (y \neq i), \quad (17)$$

$$D_x\left(\frac{d^2n_x}{d\rho^2} + \frac{1}{\rho}\frac{dn_x}{d\rho}\right) + \sum_y (z_{yx}n_y - z_{xy}n_x)n_i = 0, \quad (y \neq x); \quad (18)$$

$x = p, q, r, s$; $y = p, q, r, s, g, i$ $(z_{ix} = 0)$;

D_x, D_a, n_g constant;

$n_i' = k_i n_i$, $n_x' = k_x n_x$ at $\rho = R$, k_i, k_x being positive constants;

$n_i = n_{i0}$ at $\rho = 0$.

The quantities n are densities, as in Example 1, and the quantities D are diffusion coefficients. The coefficients z_{xy} are transition rates between the various states of excitation specified by the suffices p, q, r, s, g, i, and are integrals of the form

$$z_{xy} = au^{3/2}\int_0^\infty f_{xy}(x)(x + x_0)\exp[-bu(x + x_0)]dx, \quad (19)$$

where: a, b and x_0 are constants; the $f_{xy}(x)$ are empirical functions (excitation cross-sections, x being the energy above the threshold x_0) available in tabular form; and u is a constant, related to the mean electron energy, which has to be determined.

In this case the program is entirely self-contained; it calculates the coefficients, and assembles the matrices and vectors in the form required by the subroutines, automatically. Starting with an assumed value of u, it calculates the required coefficients z_{xy} by Simpson's rule, and assembles the appropriate matrix for solving the 4 equations (18) simultaneously (in the manner described in Section 3), for the functions n_x. These last are then used to formulate the matrix for solving equation (17), the finite-difference equations for which are arranged in the form

$$l_s n_{i,s-1} + (k_s - \lambda)n_{i,s} + m_s n_{i,s+1} = 0, \ (s = 1 \text{ to } n - 1), \quad (20)$$

in such a way that the latent root λ must be equal to 1 for the solution to be that of (17). The smallest root λ is then extracted by the latent root routine (as in Example 1, the other roots have no physical significance). This normally differs from unity, and a procedure of inverse interpolation is then used to find another value of u which yields a value of λ closer to unity after the next iteration. The whole process is repeated until convergence is obtained. With the radius R divided into 16 intervals, one whole cycle takes about 6 minutes (the integrals taking about 2 min, the simultaneous equations $2\frac{1}{2}$ min and the latent root about 1 min), and convergence is obtained to within 5 significant figures in 3 or 4 cycles, if the initial guess for u is within about 10%. (The actual program carries on from this point to evaluate the complete characteristics of the discharge from the solution so obtained.)

Example 3

In this example there are two simultaneous partial differential equations, the first of which has eigenvalues, and the second of which is non linear. If, in a gas discharge of the type described by equation (15), the gas temperature varies appreciably over the cross-section, then the single equation (15) is modified, and supplemented by a thermal-conductivity equation, giving a pair of simultaneous equations which reduce (see Appendix 2) to

$$Tf(T)\nabla^2 n + \lambda n = 0 \quad (21)$$

$$\nabla^2 T = -\frac{a\lambda n}{Tg(T)} \quad (22)$$

with $n = 0$, $T = T_w (\neq 0)$ on the boundary.

In these equations, n, T and T_w are proportional to the ion density, gas temperature, and wall temperature, respectively, the last being either a constant, or varying slowly from point to point on the wall; $f(T)$ and $g(T)$ are proportional to the effective diffusion coefficient of the ions, and to the thermal conductivity of the gas, respectively, at temperature T, and a is a constant. In most of the cases evaluated,

$$f(T) = T^{1/2} \text{ and } g(T) = T^{3/2}\left(\frac{1 + c}{T + c}\right),$$

c being a positive constant: obviously neither function can be zero at any point.

These equations have been solved for various boundaries and values of the constants, by a program which operates in the following way. The quasi-diagonal matrix A representing the Laplacian operator in (21), for the appropriate boundary, is read in. The same matrix is used for the operator in (22), a vector V being added to the right-hand sides of the finite-difference equations; the elements of V are chosen to give the correct boundary condition $T = T_w$. This vector, whose elements are zero at all the grid points except those adjacent to the boundary, is also read in, together with two vectors which are first approximations to n and T (almost any physically reasonable values are adequate for these). The program then operates as follows:

(1) The elements of the vector representing $f(T)$ are evaluated.

(2) The matrix A is multiplied row by row by the elements of $T \,.\, f(T)$.

(3) The resulting matrix is then used to evaluate the smallest latent root λ of (21), together with the corresponding latent vector n. (As in the other examples, only this root has any physical significance.)

(4) The elements of the vector representing $g(T)$ are evaluated and used to form the right-hand side vector, whose elements are $V_i - \dfrac{a\lambda n_i}{T_i g(T_i)}$, of (22).

(5) A new vector T is obtained by solving (22), using the quasi-diagonal matrix division routine.

(6) Steps (4) and (5) are repeated, using the new vector T, until the process converges.

(7) The whole calculation is repeated from (1) until final convergence is obtained.

In a typical case in Cartesian coordinates with a 6×6 grid, the first latent root converged to 4 figures in 4 iterations of the latent-root routine, the acceleration procedure being applied after the first. In subsequent cycles only two iterations, both with the acceleration procedure, were required. The iteration of equations (22) in steps (4) and (5) took longer to converge, taking 8 iterations to obtain 4 significant figures the first time round, and 6 subsequently. The whole computation converged to within 1% in 5 complete cycles and took just under 2 hours. Cases involving smaller matrices converged much more quickly; for example, with the one-dimensional matrices in polar coordinates used when there is cylindrical symmetry, convergence to 4 figures was usually obtained in 1 to 3 cycles of each iteration. (See Appendix 3 for a method of improving this program.)

In conclusion, it may be added that a variety of other equations have been solved by similar methods, including equations based on more involved operators than the Laplacian, on which these examples are based, and with more involved boundary conditions. The general scope of the method is very wide.

Acknowledgement

This work was carried out at the Research Laboratory of The British Thomson-Houston Co. Ltd. (now Associated Electrical Industries (Rugby) Ltd.), Rugby, during 1958–59.

The author would like to thank Mrs. M. L. Read, of the Engineering Mathematics Department of A.E.I. (Rugby) Ltd., who carried out the programming for Pegasus and Mercury, and also the programming and computation for most of the applications, together with his other colleagues for their helpful cooperation. He also thanks the *Journal's* referee, whose suggestions have greatly improved the paper.

References

CAYLESS, M. A. (1960). "A Generalised Theory of the Positive Column in Mercury Rare-Gas Discharges," *Proc. Fourth International Conference on Ionization Phenomena in Gases*, Uppsala, 1959, Vol. 1, p. 271: Amsterdam: North Holland Pub. Co.; "Production of Resonance Radiation in Discharge Tubes of Non-Circular Cross-Section," *Brit. J. Appl. Phys.*, Vol. 11, p. 492.

CORNOCK, A. F. (1954). "The Numerical Solution of Poisson's and the Biharmonic Equation by Matrices," *Proc. Camb. Phil. Soc.*, Vol. 50, p. 524.

DEUCE News No. 45 (December 1959). Kidsgrove: English Electric Co. Ltd.

FOX, L. (1947). "Mixed Boundary Conditions in the Relaxational Treatment of Biharmonic Problems (Plane Strain or Stress)," *Proc. Roy. Soc.*, A, Vol. 189, p. 535.

FOX, L. (1950). "The Numerical Solution of Elliptic Differential Equations when the Boundary Conditions Involve a Derivative," *Phil. Trans. Roy. Soc.*, A, Vol. 242, p. 345.

FOX, L. (1957). *The Numerical Solution of Two-Point Boundary Problems in Ordinary Differential Equations.* Oxford: The University Press.

KARLQUIST, O. (1952). "Numerical Solution of Elliptic Difference Equations by Matrix Methods," *Tellus*, Vol. 4, p. 374.

LIVESLEY, R. K. (1960). "The Analysis of Large Structural Systems," *The Computer Journal*, Vol. 3, p. 34.

N.P.L. (1957). *Modern Computing Methods*, Notes on Applied Science No. 16. London: H.M.S.O.

OSTROWSKI, A. M. (1958–59). "On the Convergence of the Rayleigh Quotient Iteration," *Arch. Rat. Mech. Anal.*, Vols. 1, p. 233; 2, p. 423; 3, pp. 325, 472; 4, p. 153.

WILKINSON, J. H. (1954). "Linear Algebra on the Pilot Ace," *Proc. Symposium on Automatic Digital Computers at N.P.L.*, p. 129. London: H.M.S.O.

[*References continued on page 60.*

WILKINSON, J. H. (1958*a*). "The Calculation of the Eigenvectors of Codiagonal Matrices," *The Computer Journal*, Vol. 1, p. 90.
WILKINSON, J. H. (1958*b*). "The Calculation of Eigenvectors by the Method of Lanczos," *The Computer Journal*, Vol. 1, p. 148.
WILKINSON, J. H. (1960). "Householder's Method for the Solution of the Algebraic Eigenproblem," *The Computer Journal*, Vol. 3, p. 23.
WILSON, L. B. (1959). "Solution of Certain Large Sets of Equations on Pegasus using Matrix Methods," *The Computer Journal*, Vol. 2, p. 130.

Appendix 1

Normal Gradient Boundary Conditions on Curved Boundaries

Finite-difference approximations involving conditions of the type $y_n' = ay + c$ on curved boundaries, where y_n is the normal gradient, have been extensively discussed by Fox (1947; 1950). In the case of second-order equations, with only first-order differences, the equations can still be expressed in terms of quasi-diagonal matrices as follows.

When $a = 0$, the method illustrated by Fig. 2(*a*) is

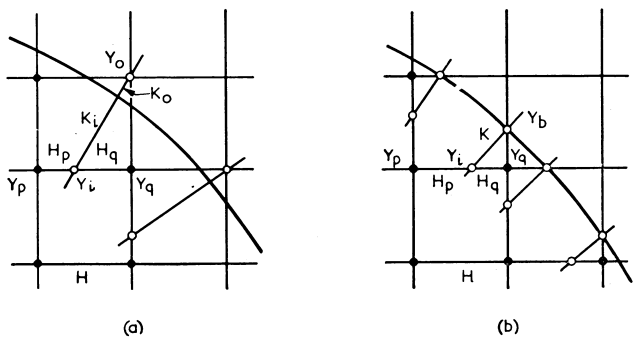


Fig. 2.—Methods for dealing with normal gradient boundary conditions

Finite difference grid points •.
Additional and interpolated points ○.

simplest. The function y is calculated at an external grid point y_0 from an interpolated value at an internal point such as y_i on the same normal, by using the boundary condition. In the notation of Fig. 2(*a*) this gives

$$y_i = \frac{1}{h}(h_p y_q + h_q y_p)$$

and

$$y_0 - y_i = c(k_0 + k_i)$$

and hence

$$y_0 = c(k_0 + k_i) + \frac{1}{h}(h_p y_q + h_q y_p).$$

This equation provides the boundary condition without destroying the quasi-diagonal arrangement, although sometimes an extra side-diagonal is introduced.

When $a \neq 0$, a method involving slightly more work, such as that illustrated by Fig. 2(*b*), is necessary. The function is calculated for points such as y_b on the boundary, again using interpolated values at internal points, giving, in the notation of Fig. 2(*b*)

$$y_b - y_i = k(ay_b + c),$$

which leads to

$$y_b = \frac{\frac{1}{h}(h_p y_q + h_q y_p) + kc}{1 - ka}.$$

This likewise provides the boundary condition without adding more than an extra side-diagonal to the matrix.

These methods are most satisfactory when the curvature of the solution is small near the boundary; this is fortunately the case with many applications. Obviously higher orders of interpolation could be used if necessary. We have never had occasion to consider the related problem for equations of higher order than the second, but similar methods are probably possible. [The biharmonic equation can be dealt with in a simpler way (Allen, 1954; Fox, 1947).]

Appendix 2

Derivation of Equations (21) and (22)

In the notation of Cayless (1960) the diffusion equation for the electrons when ionization occurs in one stage is

$$D_a \nabla^2 n_e + z_{gi} n_g n_e = 0.$$

The thermal conductivity equation is

$$K \nabla^2 T + z_e n_e n_a = 0,$$

where T is the absolute gas temperature and K the thermal conductivity. (n_e, n_g and n_a are the respective densities of the electrons, the metal vapour, and the inert gas; z_{gi} is the rate of ionization; D_a is the ambipolar diffusion coefficient, and z_e is the rate of loss of energy by the electrons in elastic collisions with the rare gas atoms.)

Now $n_a = Ap_a T^{-1}$ and $n_g = Ap_g T^{-1}$, where p_a and p_g are the respective gas pressures (constant throughout the volume of the discharge), and A is a constant. Further,

the electron density n_e is equal to the ion density n_i throughout, and the above equations become

$$TD_a\nabla^2 n_i + Az_{gi}p_a n_i = 0$$

and

$$TK\nabla^2 T + Az_e p_a n_i = 0.$$

Suitably scaled, these are equations (21) and (22) respectively. In many practical cases

$$D_a \propto T^{1/2} \text{ and } K \propto T^{3/2}\left(\frac{1+c}{T+c}\right)$$

(Sutherland's Relation), giving the forms of $f(T)$ and $g(T)$ quoted.

Appendix 3

Improved Procedure for Repeated Matrix Division

It has been pointed out to the author that if, in the elimination of the matrix elements below the diagonal, the pivotal element, and those beneath it, at any stage, are not actually altered in the computer, and if none of the elements are altered during the back-substitution process, there is some time saving by avoiding unnecessary drum transfers. Furthermore, if subsequently a new right-hand side should turn up, the new equations can be solved by the resulting matrix without making any alterations to its elements, the elimination operations being applied to the right-hand side only, a process requiring only a fraction (about $2rns$ plus ns divisions in the notation of Section 3) of the number of operations required by the complete elimination.

The elements of this matrix are, in fact, those of the lower and upper triangular factors of the original matrix [see, for example, Fox (1957)], in their respective positions, the diagonal elements of the latter being unity, and not stored.

We have not, in fact, used this procedure, but it would considerably speed programs, such as that of Example 3 of Section 5, when the same matrix is used repeatedly with successive right-hand sides. Other uses would be in the latent-root routine (when used without the acceleration process) or when calculating the "difference correction."