OAK RIDGE NATIONAL LABORATORY
operated by
UNION CARBIDE CORPORATION
NUCLEAR DIVISION
for the
U.S. ATOMIC ENERGY COMMISSION

ORNL - TM - 1933

COPY NO. 133

DATE - August 30, 1967

"MATEXP," A GENERAL PURPOSE DIGITAL COMPUTER PROGRAM FOR SOLVING ORDINARY DIFFERENTIAL EQUATIONS BY THE MATRIX EXPONENTIAL METHOD

S. J. Ball R. K. Adams

ABSTRACT

MATEXP, a general purpose digital computer program, was written for solving systems of ordinary differential equations by the matrix exponential method. MATEXP has several advantages over standard numerical integration routines. It gives virtually exact solutions to constant-coefficient homogeneous equations and to nonhomogeneous equations for which the forcing functions are constant during the computation interval. The speed at which the equations are solved and the accuracy of the solution are essentially unaffected either by the degree of cross-coupling of the equations or by whether or not the coefficient matrix is nonsingular or that its eigenvalues are distinct.

The method has been extended to nonlinear equations and equations with time-varying coefficients; this use is very effective for engineering systems analysis problems.

CONTENTS

1. INTRODUCTION

The matrix exponential method of solving differential equations was first described to the authors by Prof. Henry Paynter of MIT, who with his students[1-3] developed this method into a practical engineering tool. The basic technique was derived many years ago,[4] and even then it was an elegant method of obtaining exact solutions for a set of constant coefficient, homogeneous differential equations. The matrix exponential technique is ideally suited to digital computation and is very simple to implement, especially when compared with most quadrature methods.

Only two persons besides Prof. Paynter have done extensive work in this area. L. Pease[5] of Atomic Energy of Canada, Ltd., independently developed the method simultaneously with Paynter. The work of Paynter and Pease formed the basis for our implementation and, perhaps, refinement of the method, although the work of several researchers[5-9] established the rigor of the central technique.

---

[1]J. Suez, Automated Programming for Analog Computers, M.S. thesis, MIT, Aug. 1962.

[2]H.C.H. Lee, Some Finite Difference Models for Linear and Nonlinear Control Studies Using Digital Computation, M.S. thesis, MIT, Aug. 1962.

[3]H. M. Paynter and J. Suez, "Automatic Digital Setup and Scaling of Analog Computers," Trans. ISA, 3, 55-64 (Jan. 1964).

[4]E. Artin, from O. Schreier and E. Sperner, Introduction to Modern Algebra and Matrix Theory (1935); Translated from German, Chelsea Publ. Co., N.Y., 1951, pp. 319-320.

[5]L. Pease, DEEMS, A Fortran Program for Solving the First-Degree Coupled Differential Equations by Expansion in Matrix Series, AECL-1898 (Oct. 1963, reprinted Feb. 1964).

[6]E. G. Keller, Mathematics of Modern Engineering, vol II, Mathematical Engineering, Wiley, N.Y., 1942, pp. 234-246.

[7]R. Bellman, Introduction to Matrix Analysis, McGraw-Hill, N.Y., 1960, pp. 165-173.

More recently, M. L. Liou of Bell Telephone Laboratories made important contributions to the matrix exponential method.[10,11]

Because this method can give virtually exact[12] solutions to systems of equations, it is of considerable interest to most engineers engaged in systems analysis, automatic control, and simulation. Also, systems engineers have long recognized that one essential difference between the analog computer and the digital computer is the awkward (at best) manner in which the digital machine can perform integration. The matrix exponential method, on the other hand, requires the digital computer to perform mainly matrix manipulations, which it can do in a very straightforward and efficient manner.

The matrix exponential techniques have worked well for a large general class of simulation problems which constitute the bulk of the work in the systems analysis and automatic control fields. Indeed, by use of the methods described in Sect. 3.4, certain types of nonlinear equations can be solved as a natural extension of the basic matrix exponential method.

---

[8] F. R. Gantmakher, Applications of the Theory of Matrices, Interscience, N.Y., 1959, pp. 135-9 (translation of Russian original book: Theory of Matrices, 1954).

[9] L. A. Pipes, Applied Mathematics for Engineers and Physicists, 2d ed., McGraw-Hill, N.Y., 1958, pp. 101-4.

[10] M. L. Liou, "A Novel Method of Evaluating Transient Responses," Proc. IEEE, 54 (1), 20-23 (Jan. 1966).

[11] F. F. Kuo and J. F. Kaiser, eds., System Analysis by Digital Computer, Wiley, N.Y., 1966, pp. 99-129.

[12] "Virtually exact" means that the solution can be calculated to as great a precision as is desired, consistent with the precision obtainable with a given computer word length. In other words, the precision of the method is not necessarily limited by the convergence of any approximate quadrature (integration) formula, simply because quadrature is not performed.

The matrix exponential method has also been implemented and used extensively in Fourier analysis problems by simulating band-pass filters.[13,14] Instead of calculating correlation functions (and subsequently their Fourier transforms) digital filtering can be used to obtain spectral density estimates and transfer functions from noise data. Calculations using filtering techniques are of comparable accuracy and typically more efficient than the conventional methods.

MATEXP has also been used in a special technique to calculate the sensitivities of the time response of a system to changes in parameter values.[15] A description of a subroutine which was written to implement time response sensitivity calculations is given in Sect. 5.2.3.

MATEXP has been developed and modified over a period of several years, and its present form reflects the considerable number of helpful suggestions we have had from many people. We are particularly grateful to Prof. H. M. Paynter for first introducing us to the method, and to Prof. T. W. Kerlin of the University of Tennessee, and J. V. Wilson of ORNL for their help and encouragement.

2. DEVELOPMENT OF THE MATRIX EXPONENTIAL METHOD

2.1 For Homogeneous Equations

Consider the first-order scalar, linear, homogeneous differential equation (with constant coefficient)

$$\frac{dx}{dt} + ax = 0 , \qquad (1)$$

---

[13] S. J. Ball, A Digital Filtering Technique for Efficient Fourier Transform Calculations, ORNL-TM-1778 (July 1967).

[14] T. W. Kerlin and S. J. Ball, Experimental Dynamic Analysis of the Molten-Salt Reactor Experiment, ORNL-TM-1647 (Oct. 1966).

[15] T. W. Kerlin, "Sensitivities by the State Variable Method," Simulation, 8(6), 337-345 (June 1967).

whose solution is

$$x = e^{-at} x_0 . \tag{2}$$

An interesting characteristic of the solution is that, for any time interval τ, the value of x at the end of the interval is a product of an exponential term $e^{-a\tau}$ and the value of x at the beginning of the interval, i.e.

$$x_{t+\tau} = e^{-a\tau} x_t . \tag{3}$$

This will be referred to as the "incremental solution."

Now because a system of homogeneous linear equations of any order can always be broken up into a set of first-order equations, consider the following set of equations

$$\begin{aligned} \frac{dx_1}{dt} &= a_{11} x_1 + a_{12} x_2 + \cdots a_{1n} x_n , \\ \frac{dx_2}{dt} &= a_{21} x_1 + a_{22} x_2 + \cdots a_{2n} x_n , \\ &\vdots \\ \frac{dx_n}{dt} &= a_{n1} x_1 + a_{n2} x_2 + \cdots a_{nn} x_n . \end{aligned} \tag{4}$$

This array can be expressed compactly in matrix form as a first-order, linear, homogeneous, matrix differential equation with constant coefficients, i.e.

$$\frac{dX}{dt} = AX , \tag{5}$$

where X is the column vector of state variables x_i

$$X \equiv \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

and A represents the coefficient matrix

$$A \equiv \begin{pmatrix} a_{11} & a_{12} & \cdots\cdots & a_{1n} \\ a_{21} & a_{22} & \cdots\cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots\cdots & a_{nn} \end{pmatrix}$$

This matrix equation has the solution

$$X_t = \epsilon^{At} X_0 \ . \tag{6}$$

For a formal proof that Eq. (6) is the desired solution, the reader is referred to Bellman.[7] However, the following simple proof is somewhat less formal. First, if $dX/dt = AX$, then $\frac{d^2X}{dt^2} = A \frac{dX}{dt} =$ $A\ A\ X = A^2\ X$; similarly, $\frac{d^3X}{dt^3} = A^3\ X$, so that $\frac{d^mX}{dt^m} = A^m\ X$. (7)

If X_t is expanded about zero in a Taylor's series,

$$X_t = X_0 + \frac{t}{1!} \frac{dX}{dt}\bigg|_{t=0} + \frac{t^2}{2!} \frac{d^2X}{dt^2}\bigg|_{t=0} + \cdots\cdot \frac{t^m}{m!} \frac{d^mX}{dt^m}\bigg|_{t=0}$$

With Eq. (7) substituted for the derivative,

$$X_t = \left(I + \frac{At}{1!} + \frac{A^2t^2}{2!} + \cdots\cdot \right) X_0$$

or

$$X_t = \epsilon^{At} X_0 \quad \text{(Q.E.D.)} \tag{8}$$

The "incremental solution" is

$$X_{t+\tau} = \epsilon^{A\tau} X_t \ , \tag{9}$$

where $\epsilon^{A\tau}$, the matrix exponential, is defined analogously to the scalar exponential as

$$\epsilon^{A\tau} = I + A\tau + \frac{(A\tau)^2}{2!} + \frac{(A\tau)^3}{3!} + \ldots \frac{(A\tau)^k}{k!} \tag{10}$$

in which I is the identity matrix

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

2.2 For Nonhomogeneous Equations

The matrix equation representing a system of first-order, constant coefficient differential equations with nonzero forcing functions is the nonhomogeneous equation

$$\frac{dX}{dt} = AX + Z, \tag{11}$$

where Z is the disturbance, or forcing function, vector.

A general incremental solution of the nonhomogeneous equation as derived by Liou[11] is

$$X_{t+\tau} = \epsilon^{A\tau} X_t + \epsilon^{A(t+\tau)} \int_t^{t+\tau} \epsilon^{-A\tau} Z_\tau \, d\tau \, . \tag{12}$$

An exact solution derived from Eq. (12) for the case where the forcing function Z is constant over the interval t to t+τ is

$$X_{t+\tau} = \epsilon^{A\tau} X_t + (\epsilon^{A\tau} - I) A^{-1} Z_t \, . \tag{13}$$

It is important to note that the inverse of A need not be calculated to evaluate Eq. (13) since

$$(\epsilon^{A\tau} - I)A^{-1} = \left[\cancel{I} + A\tau + \frac{(A\tau)^2}{2!} + \cdots \frac{(A\tau)^k}{k!} - \cancel{I} \right] A^{-1} \, ,$$

$$= I\tau + \frac{A\tau^2}{2!} + \frac{A^2\tau^3}{3!} + \cdots \frac{A^{k-1}\tau^k}{k!} \, ,$$

$$= \tau \left[I + \frac{A\tau}{2!} + \frac{(A\tau)^2}{3!} + \ldots \frac{(A\tau)^{k-1}}{k!} \right],$$

$$= \tau \sum_{k=1}^{\infty} \frac{(A\tau)^{k-1}}{k!} \quad . \tag{14}$$

Because this series is similar to that used to represent $\epsilon^{A\tau}$, the computer program can calculate the two required matrices concurrently, since the kth term of the $(\epsilon^{A\tau}-I)A^{-1}$ series equals the (k-1)th term of the $\epsilon^{A\tau}$ series times (τ/k). In the MATEXP program, the $\epsilon^{A\tau}$ matrix is called the "C" matrix and the $(\epsilon^{A\tau}-I)A^{-1}$ matrix is called the "HP" matrix (in honor of H. Paynter).

At this point, two essential features of the matrix exponential method are emphasized:

1. The exponential matrices can be computed by the series approximation to nearly any desired precision (typically, 1 part in 10^6 is specified for MATEXP calculations). Hence, for homogeneous equations and for nonhomogeneous equations in which the forcing functions remain constant over the computation time interval, the solutions are virtually exact solutions.
2. The solution vector can be updated successively by a time increment τ by two matrix multiplications:

$$X_{\tau} = C\, X_0 + HP\, Z_0$$

$$X_{2\tau} = C\, X_{\tau} + HP\, Z_{\tau}$$

$$\vdots$$

etc

If it is assumed that just one time increment value τ is required, the C and HP matrices need to be evaluated only once.

An exact solution to the set of nonhomogeneous differential equations can also be derived from Eq. (12) for the case where the forcing function Z varies linearly within the computation interval τ. In terms of the matrix exponential series approximations, the

trapezoid forcing function incremental solution is

$$X_{t+\tau} = \epsilon^{A\tau} X_t + \tau \sum_{k=1}^{\infty} \left(\frac{1}{k!} - \frac{1}{(k+1)!} \right) (A\tau)^{k-1} Z_t + \tau \sum_{k=1}^{\infty} \frac{(A\tau)^{k-1}}{(k+1)!} Z_{t+\tau} \cdot \qquad (15)$$

Liou[11] has also developed a recursive formula for accurate approximations of continuous forcing functions which uses a Simpson's rule approximation of the nonhomogeneous solution, Eq. (12), within the time interval τ:

$$X_{t+\tau} \approx \epsilon^{A\tau} \left[X_t + \frac{\tau}{6} Z_t \right] + \frac{2\tau}{3} \epsilon^{A\tau/2} Z_{t+\tau/2} + \frac{\tau}{6} Z_{t+\tau} \cdot \qquad (16)$$

As with the case of the step-wise varying forcing functions, the matrices required for Eqs. (15) and (16) need to be evaluated just once at the start. These features are not presently included in the MATEXP code, but could readily be added as options.

2.3 Miscellaneous Features of the Matrix Exponential

Since the matrix exponential principle has been a part of the mathematical literature for many years, the matrix exponential has had at least two other names: the fundamental matrix, and the transition matrix. Besides the series approximation method, an analytical method is often used to calculate this matrix;[9] however, the eigenvalues of A and their eigenvectors must be calculated and the initial condition vector must be transformed by a matrix comprised of the eigenvectors. It is emphasized that the series method used in MATEXP does not require that the coefficient matrix be nonsingular (i.e., have a nonzero determinant) or that its eigenvalues be distinct (a case where the analytical solution has terms of the form te^{bt} and cannot be expressed as the sum of exponentials). The latter condition, which occurs in problems where two time constants in a decay chain are equal, was one of

the problems that Pease encountered in reactor burnup calculations that prompted him to develop the matrix exponential method.[5]

Another feature noted by Pease (but not included in MATEXP) is that the average solution vector $\overline{X}$ could be obtained directly from a matrix exponential type calculation.

From the mean value theorem,

$$\overline{X} = \frac{1}{\tau}\int_0^\tau X_t \, dt,$$

$\overline{X}$ can be obtained by integrating the equation for X in terms of C and HP:

$$\overline{X} = \frac{1}{\tau}\int_0^\tau X_t \, dt = \frac{1}{\tau}\int_0^\tau \left[C\, X_0 + (HP)\, Z_0 \right] dt \, . \qquad (17)$$

Term by term integration of the series approximations for C and HP gives

$$\int_0^\tau C \, dt = \tau\left[I + \frac{A\tau}{2!} + \frac{(A\tau)^2}{3!} + \frac{(A\tau)^3}{4!} + \ldots \right] \equiv HP \, , \qquad (18)$$

and

$$\int_0^\tau HP \, dt = \tau^2\left[\frac{I}{2!} + \frac{A\tau}{3!} + \frac{(A\tau)^2}{4!} + \ldots \right] . \qquad (19)$$

The latter series, like the HP matrix calculation, could easily be made concurrent with the other matrix exponential calculations.

The accuracy of MATEXP solutions, both in absolute terms and compared with other methods, is difficult to estimate quantitatively for the general case. Even for those cases that are solved "exactly," the successive multiplications of the solution vector by the matrix exponential naturally tend to accumulate errors. However, with precise calculations of the C and HP matrices as recommended in the Appendix, Sect. 5.1, test cases have shown this error to be negligible for large systems (40 x 40), even after many thousands of updating calculations. Liou[11] has developed an alternative method of evaluating the C and HP matrices to a prescribed accuracy.

The nature of the matrix exponential method permits the use of

much larger computation time intervals τ than would be feasible for most numerical integration solutions. For constant-coefficient equations and a given τ, it would be safe to assume that MATEXP would be inherently more accurate. As is usually the case, however, it would be unwise to generalize about nonlinear equations. Nonlinear solutions are discussed further in Sect. 3.4.

Eq. (20) gives a rough estimate of MATEXP solution times on the IBM-7090 computer, assuming that a negligible time is spent in the peripheral subroutines:

$$\text{Solution time(min)} \approx 3.0 \times 10^{-6} (\text{NE})^2 \text{ NT} , \qquad (20)$$

where NE is the number of equations, and NT is the number of computation time intervals. For example, a 59 x 59 system run for 1000 time steps took 10 min, and an 8 x 8 run for 10,000 steps took 1.5 min. The solution time factor will vary from about 2×10^{-6} to 7×10^{-6}, depending on the amount of extra subroutine computation and printout, and will be approximately halved for homogeneous equations.

The present "standard" version of the MATEXP program solves up to 60th-order equations and uses about 22,000 words of core storage. In a 32,000 word computer, the extra 10,000 words can be used for special programming or storage, or the order of the equation can be increased to about 80. Since, for larger problems, tape or other slower storage devices would be required to calculate the matrix exponential functions, the overall efficiency of the method would be reduced.

Two other interesting, though perhaps purely academic, features of the matrix exponential technique are that the solution time increment can be negative (allowing one to go backwards) and that the A matrix can contain complex coefficients.

3. DESCRIPTION OF MATEXP PROGRAM AND OPTIONS

3.1 Basic Input Information

The MATEXP program was written with the intent that it should be easy to use for a wide variety of differential equation problems.

Unfortunately, as a program becomes more general, i.e. the more options and special features the program has, it becomes more difficult to explain the program and to use it for any given problem. Consequently, any apparent awkwardness and complications in the following discussion are due to a desire to make it general, and any omissions are due to a desire to keep it simple.

The basic parts of the code are: the main program, MATEXP; the utility subroutine used for outputting, OUTPUT; and the subroutine for calculating forcing(or disturbance)functions, DISTRB. To solve linear, constant-coefficient differential equations that are homogeneous (i.e. have no forcing functions) or which have only fixed forcing functions, all the required data can be read in and no extra programming is necessary. For equations of the form

$$\frac{dX}{dt} = AX + Z \ ,$$

the initial values of the X vector, the coefficient matrix A, and the (fixed) disturbance vector Z may be read in. Other information required for each run is the following:

1. number of equations,
2. initial time (or other independent variable),
3. computation time interval,
4. final time,
5. interval at which solution vector X and disturbance vector Z are to be printed.

Since many elements of the coefficient matrix A are often zero, only the nonzero elements need to be read in. This makes it necessary to identify each coefficient with its row and column number. The nonzero values of the initial condition and fixed disturbance vectors, with their row numbers, are read in similarly.

Since successive runs might require no changes (or only a few) in input data from the previous run, options are provided so that only the altered data has to be read in.

An option is also available whereby the last value of the X vector from one run can be used as the starting value of the succeeding run.

This option can be used if changes in the computation or printing interval are required in the middle of a solution or if certain iteration or successive approximation schemes are being used.

A complete description of the inputs and options is given in the Appendix, Sect. 5.

3.2 Alternative Methods of Generating the Coefficient Matrix A

Although the most straightforward method of inputting the coefficient matrix is to read it in, very often it is advantageous to have some or all of the elements calculated from system parameter values. One option of MATEXP provides for this to be done by special programming on the first call of DISTRB. An alternative is to use an "algebra table" routine developed by Kerlin and Lucius.[16] This routine calculates the matrix elements from input parameter values without any special programming. The general expression used for calculating an element a_{ij} in terms of parameters P_k and their exponents $E_{k\ell}$ is

$$a_{ij} = C_1 P_1^{E_{11}} P_2^{E_{21}} P_3^{E_{31}} \cdots P_n^{E_{n1}} + C_2 P_1^{E_{12}} P_2^{E_{22}} P_3^{E_{32}} \cdots P_n^{E_{n2}} + \cdots$$

or

$$a_{ij} = \sum_{\ell=1}^{m} C_\ell \prod_{k=1}^{n} P^{E_{k\ell}} \qquad (21)$$

A complete description of the program is given in reference 16.

Beside the fact that it is sometimes convenient to have the coefficient matrix calculated by the computer, in some cases computer computation is almost necessary to obtain accurate solutions. This was the case for one reactor dynamics calculation where the coefficients were first carefully calculated on a 20-in. slide rule, then by the machine. The difference in the steady-state solution for neutron

---

[16] T. W. Kerlin and J. L. Lucius, A Technique for Calculating Frequency Response and its Sensitivity to Parameter Changes for Multi-Variable Systems, ORNL-TM-1189 (June 1965).

level after a reactivity insertion was approximately a factor of 2.

3.3 Alternative Methods of Generating the Forcing Function Vector Z

When variable forcing functions are needed, a special program must usually be written and included in DISTRB. Two special forcing function subroutines have been written to simplify the programming: DFG, for approximating arbitrary functions; and TRLG, for approximating variable transport lags. They are both described in Sect. 3.5.

For cases where the forcing function is a solution to an ordinary differential equation, this equation can simply be added to the system matrix, and an exact solution can be obtained. As an example, assume that a sinusoidal forcing function is used to excite a damped spring-mass system. The quadratic equation that describes the displacement y of the mass with time is

$$\frac{d^2y}{dt^2} + a\frac{dy}{dt} + by = c \sin(\omega t + \phi) , \qquad (22)$$

where ω is the frequency of the sinusoidal input (radians/time). To arrange the equation in terms of first-order derivatives, let

$$x_1 \equiv \frac{dy}{dt} , \qquad (23)$$

$$x_2 \equiv y . \qquad (24)$$

Solving for d^2y/dt^2 (or dx_1/dt), we obtain

$$\frac{dx_1}{dt} = -ax_1 - bx_2 + c \sin(\omega t + \phi), \qquad (25)$$

and

$$\frac{dx_2}{dt} = x_1 . \qquad (26)$$

The equation for a pure oscillator with frequency ω is

$$\frac{d^2s}{dt^2} + \omega^2 s = 0. \qquad (27)$$

If we let $x_3 = \frac{ds}{dt}$, and $x_4 = \omega s$, then

$$\frac{dx_3}{dt} = -\omega x_4 \ , \tag{28}$$

$$\frac{dx_4}{dt} = \omega \, x_3 \ . \tag{29}$$

If the initial conditions of x_3 and x_4 are zero and -1, respectively, then

$$x_3(t) = \sin \omega t \ , \tag{30}$$

$$x_4(t) = -\cos \omega t \ . \tag{31}$$

Thus cx_3 could be substituted for c sin $(\omega t + \phi)$ in Eq.(25). The required initial conditions of velocity $x_1(0)$ and displacement $x_2(0)$ must also be specified.

The coefficient matrix for this example is

$$A = \begin{pmatrix} -a & -b & +c & 0 \\ +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\omega \\ 0 & 0 & +\omega & 0 \end{pmatrix}$$

If the sinusoidal input were introduced as a forcing function, it would appear as a stair-step approximation of a sine wave, and the accuracy of the solution would depend on the accuracy of this approximation. A comparison of the approximate and exact solutions for a specific example is shown in Fig. 1. In the approximate solution, a first-order extrapolation was used to approximate the average value of the forcing function over the time interval.

In this example, the system has a natural frequency of 1.0 radian/sec and a damping factor of 0.25, and the driving sinusoid has a frequency of 2.0 radians/sec. The computation interval of 0.5 sec for the approximate case gives about seven computations per cycle of the driving function. Figure 1 also shows the response after a long time where the excellent stability and accuracy of both

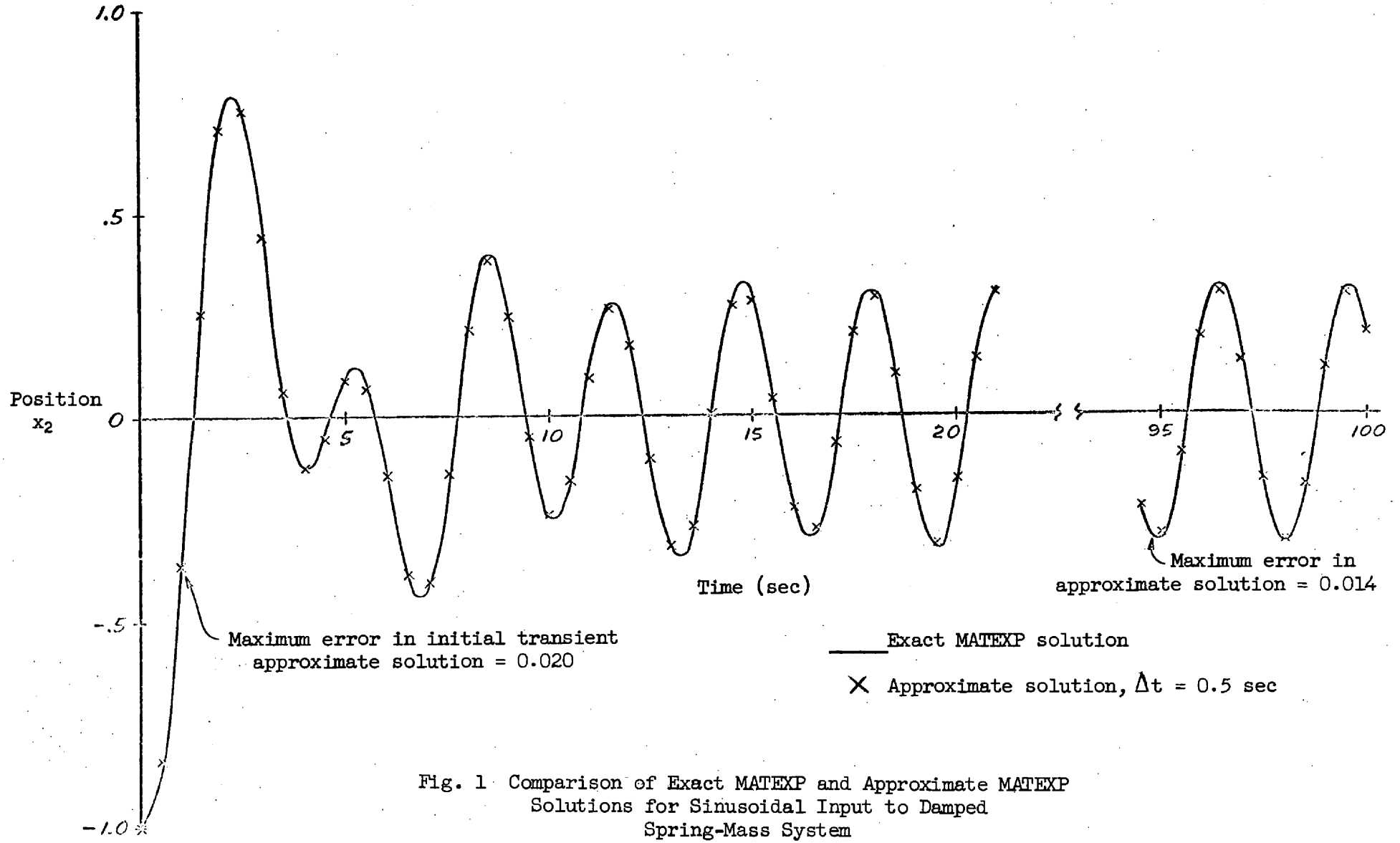


Fig. 1 Comparison of Exact MATEXP and Approximate MATEXP Solutions for Sinusoidal Input to Damped Spring-Mass System

solutions can be seen. This type of calculation is, historically, very difficult to do with standard digital methods.[17]

3.4 Methods for Solving Time-Varying-Parameter and Nonlinear Differential Equations

It was shown in Sect. 2 that the MATEXP method can provide exact solutions to sets of constant-coefficient, homogeneous differential equations and to nonhomogeneous equations for which the forcing functions can be represented by stepwise-varying functions. Since forcing functions are usually smoothly varying, the accuracy of the solution would naturally depend on the accuracy of the stair-step approximations.

Likewise, in the case of time-varying-parameter, or nonlinear, equations, the variations in the coefficient matrix A can be approximated by stepwise variations. For a variable A matrix, however, the matrix exponentials (C and HP) would both have to be re-evaluated at each computation interval. Although this may still be an efficient method for low-order equations (~10 or less), it could be quite time consuming for larger problems.

A more efficient method of solution is to modify, or "fudge," the forcing function vector so that it compensates for the variation in coefficients while the A, C, and HP matrices remain constant. This is shown schematically in Fig. 2.

---

[17] R. A. Gaskill, "Fact and Fallacy in Digital Simulation," Simulation, 5 (5), 309-313 (Nov. 1965).

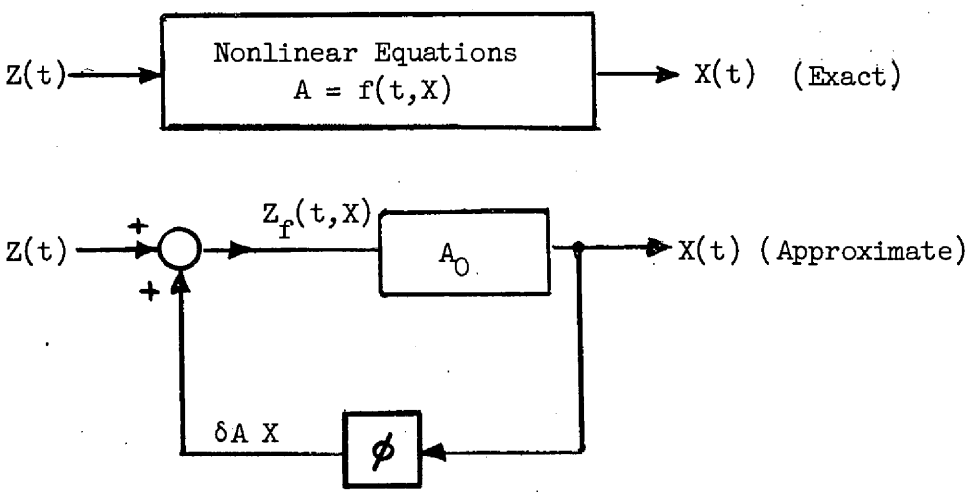


Fig. 2. Approximate Solution Using Fudged Forcing Functions.

Each component of the fudged forcing-function vector is calculated by adding all the coefficient perturbation quantities in the row. For example, assume one row of the matrix equation is

$$\frac{dx_1}{dt} = a_{11}(t)\, x_1 + a_{12}\, x_2 + a_{13}(t)\, x_3 + z_1(t) ,$$

where a_{11}, a_{13}, and z_1 are variables and a_{12} is a constant.

Let

$$a_{11}(t) \equiv (a_{11})_0 + a'_{11} ,$$

and

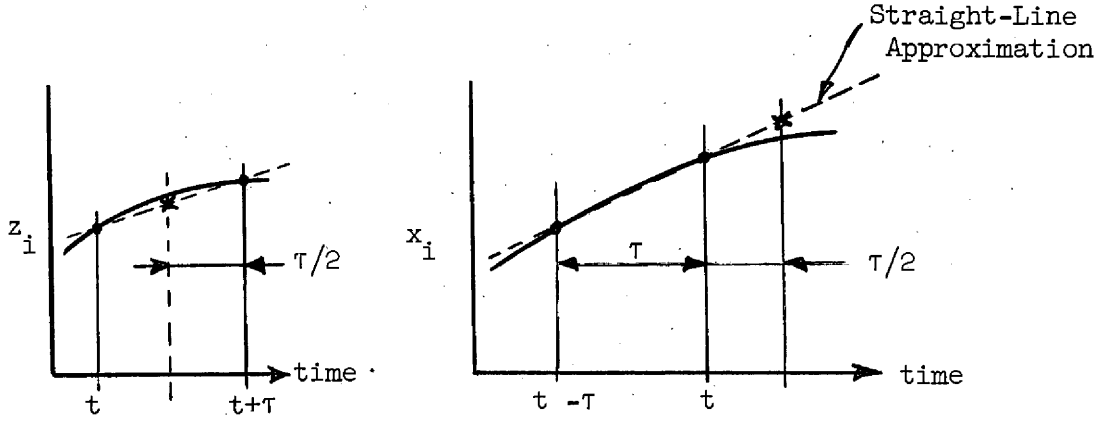
$$a_{13}(t) \equiv (a_{13})_0 + a'_{13} .$$

Then the equation can be rewritten

$$\frac{dx_1}{dt} = (a_{11})_0\, x_1 + a_{12}\, x_2 + (a_{13})_0\, x_3 + \underbrace{z_1(t) + a'_{11}\, x_1 + a'_{13}\, x_3}_{\equiv z_f(t,x)} .$$

Again, the forcing function z_f would actually be smoothly varying, but in the MATEXP difference equations, it is approximated by a stair-step function.

For the case where the coefficients and/or the forcing functions are known functions of time, much greater accuracy (for a given computation interval τ) results from using approximate mean values, rather than initial values, of the functions in the computation interval. First-order approximations of the mean values can be obtained by evaluating the time-varying forcing functions and matrix elements at $(t + \tau/2)$ instead of at (t). First-order extrapolations of the mean values of the solution vector X should also be used where coefficients are functions of X, as shown in Fig. 3.



$$x_i(t+\frac{\tau}{2}) \approx x_i(t) + \frac{x_i(t) - x_i(t-\tau)}{2}$$

Fig. 3. First-Order Extrapolation of Mean Values of z and x at $(t+\frac{\tau}{2})$.

The use of an auxiliary subroutine VARCO greatly simplifies the programming required to use first-order extrapolation calculations to find approximate mean values of the forcing function. VARCO is described in detail in Sect. 5.2.

The only way of guaranteeing that the solution is accurate is to reduce the computation interval τ until further reductions make no significant difference in the solution. A simple, intuitive estimation

of the accuracy, however, may be obtained by noting the maximum amount of change in the solution and coefficient values within a computation interval. If these changes are only a few percent of the values of the functions at the start of the interval, then the first-order approximations will probably give very accurate answers. The true accuracy of the representation of a nonlinearity should also be considered when trying to "squeeze" too much accuracy out of a solution.

The use of fudged forcing functions for the solution of nonlinear differential equations is very effective when relatively few of the matrix coefficients are variable. In this case one might consider the linear portion of the system of equations as being solved by an extremely accurate analog computer, while the nonlinear portion is simulated by a not-quite-so-accurate computer. If most of the matrix coefficients are variable, then the more conventional numerical solution methods might be more practical than MATEXP.

More detailed discussions of the theory and use of fudged forcing functions have been found disguised in sophisticated mathematical treatises by Wolf[18] and Frazer et al.[19]

3.5 Special Forcing Function Subroutines

Since special programming is required in the DISTRB subroutine to generate variable forcing functions for the differential equations, two general purpose subroutines were written to facilitate this programming for some problems.

3.5.1 Arbitrary Function Generation - DFG

The arbitrary function generation subroutine DFG provides a means of generating approximations of single-valued functions of one variable where the arbitrary function curve is represented by a

---

[18] A. A. Wolf, "Some Recent Advances in the Analysis and Synthesis of Nonlinear Systems", Am. Inst. Elec. Engrs. transactions paper No. 61-713.

[19] R. A. Frazer, W. J. Duncan, and A. R. Collar, Elementary Matrices, Cambridge University Press, 1957, pp. 232-45.

series of linear segments (Fig. 4). The principle is identical to that of the diode function generator (hence DFG) used in analog computation.

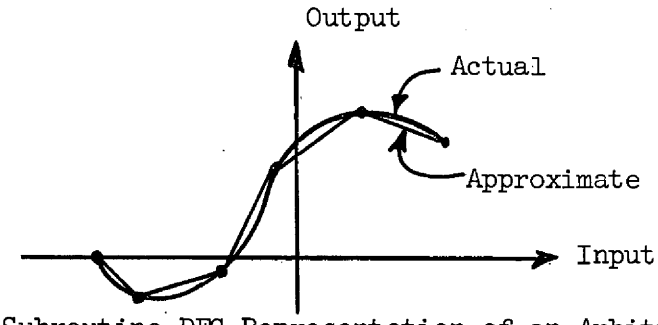


Fig. 4. Subroutine DFG Representation of an Arbitrary Function of One Variable.

DFG in its standard form arbitrarily allows for up to 8 functions with up to 32 points (or 31 line segments) per function. Inputs required are the ordinate and abscissa values of the line-segment end points. If more functions or finer approximations are required, the dimensions could be changed easily. More details on the program and a Fortran listing are given in the Appendix, Sect. 5.

3.5.2 Variable Transport Lag Generation - TRLG

A transport lag (also known as a pure time delay, or dead time) actually represents a distributed parameter system; hence, its representation in a lumped-parameter solution will be only approximate. The output z from a pure delay device with an input x and a fixed delay time τ is

$$z(t) = x\ (t - \tau).$$

If τ is variable, then the relationship between z and x is a function of the time history of τ.

The variable time-delay problem is best illustrated by fluid flow in a pipe where the inlet temperature and flow rate are both variable. The assumptions required for a pure delay are:

1. there is no heat transfer to the pipe;
2. the fluid density is constant;
3. plug flow exists, i.e., there is no mixing of the fluid in the direction of flow.

The technique used in TRLG is to sample the inlet temperature x and the flow rate W at each computation time interval T, thereby keeping an inventory on each slug of fluid in the pipe. The total weight of fluid in the pipe is computed from the initial transport time τ_i and the flow rate W_i:

$$P_{total}\ (lb) = W_i\ (lb/sec) \times \tau_i\ (sec)\ .$$

Similarly, the weight of fluid that enters during each time interval T is W(t) x T. Since the fluid density is constant, the weight of fluid that leaves during that interval T is equal to the weight of the inlet slug.

As an example, assume that the temperature profile in the pipe is as shown in Fig. 5 and the slug at the inlet of ΔP_0 lb is about to enter. The slug at the outlet is ΔP_n at a temperature x_n, where $\Delta P_n > \Delta P_0$. When ΔP_0 enters, the outlet slug temperature will be equal to x_n, and the whole profile will be shifted to the right by ΔP_0 lb. The weight of the new slug just upstream of the exit is then $(\Delta P_n - \Delta P_0)$.

If ΔP_0 had been greater than ΔP_n, the outlet slug would have taken as much of the upstream inventory (i.e., ΔP_{n-1}, ΔP_{n-2}, etc.) as required (up to 300 samples), and the outlet slug temperature z would be computed as the weighted average of the slug temperatures. For example

if

$$\Delta P_0 = \Delta P_n + 0.5\ \Delta P_{n-1}\ ,$$

then

$$z = \frac{\Delta P_n\ x_n + 0.5\ \Delta P_{n-1}\ x_{n-1}}{\Delta P_n + 0.5\ \Delta P_{n-1}}\ .$$

If the maximum delay time (minimum flow rate) would use up too many storage locations, the sampling would be done every other (or every third, etc.) computation interval. With a variable lag, a minimum expected flow rate must be specified to calculate how often to sample.

The input variables supplied by the calling program for each call of TRLG are XT (e.g., fluid temperatures) and the flow rates W (in

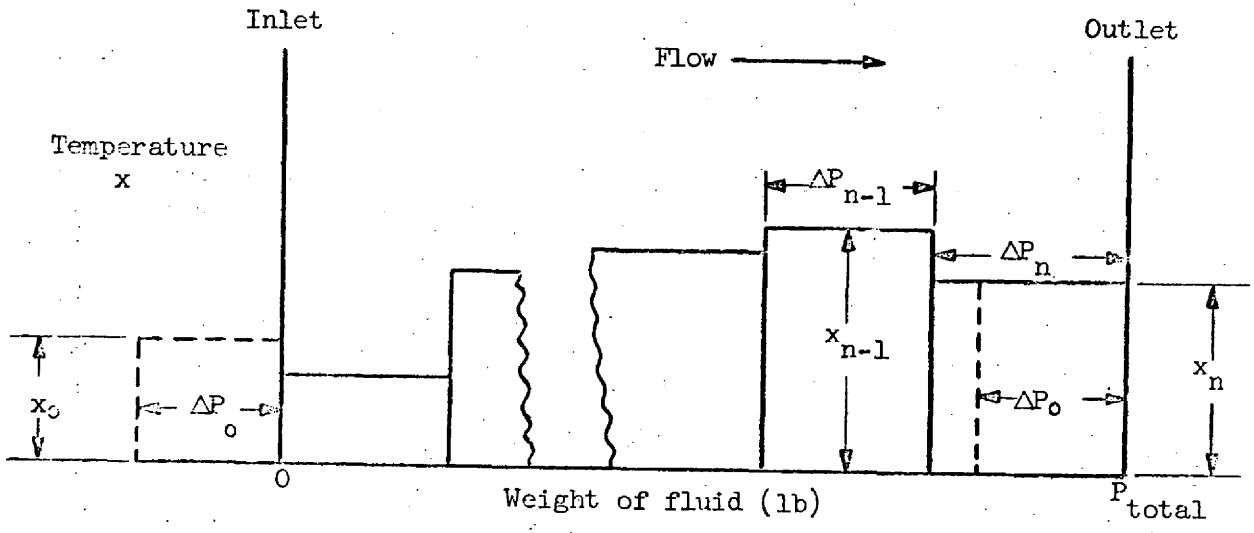


Fig. 5. Temperature Profile of Fluid in Pipe.

terms of mass/time, unity for full flow, or some percentage of full scale). The lagged functions ZT are returned by TRLG.

On the first call of TRLG, the flag NI should be zero, and the following input data are read in:

NLAGS = number of functions used,

TI = initial values of transport lag time for each function,

WMIN = minimum expected values of flow W for each function.

The initial values of fluid temperatures in the pipes are set equal to the initial values of inlet temperatures. If specific initial temperature profiles are required, they can be read in with only a minor change being required in the program. The standard version of TRLG provides for up to six lags with up to 300 samples per lag. If more or fewer lags or points are desired, the statements labeled DIMENS in the comment field can be changed accordingly.

More details on TRLG and a Fortran listing are in the Appendix, Sect. 5.

There are two other techniques that are commonly used to represent transport delays:

1. A series of n first-order lags, or "well-stirred tanks," with time constants τ/n;
2. A Padé approximation,[20] which uses several terms of a series approximation of $\epsilon^{-\tau S}$ (the Laplacian representation of a pure delay), where S is the Laplacian argument.

Both the series lag and Padé methods have accuracy and flexibility limitations that would be prohibitive for certain problems.[21] Since the digital computer is quite proficient at sampling data, the sampled data approximation as used in the TRLG subroutine is recommended as the most efficient and accurate method.

---

[20] A. E. Rogers and T. W. Connolly, Analog Computation in Engineering Design, McGraw-Hill, N.Y., 1960, pp. 419-24.

[21] S. G. Margolis and J. J. O'Donnell, "Rigorous Treatment of Variable Time Delays", IEEE Trans. on Electronic Computers, Vol. EC-12, June 1963, pp 307-9.

4. SUMMARY AND CONCLUSIONS

The matrix exponential method has a number of advantages over the more common integration schemes for a large and significant class of ordinary differential equation problems. The speed and accuracy of MATEXP have the potential of reducing computing costs for large problems and of making more "real-time" computations feasible for on-line digital computation, control, and optimization calculations.

The MATEXP program has been developed over a period of several years, mainly through use in simulation problems. There are, however, at least three other areas in which the matrix exponential method might be effective:

1. Automatic parameter estimation - where the parameters of the model differential equations are adjusted to optimize the agreement between theoretical and experimental response curves. A computer program to implement this technique is currently under development;
2. Solution of nonlinear algebraic equations by the method of steepest ascents; and
3. Boundary value problems.

Other refinements that have been used with the MATEXP code include the addition of an automatic plotting subroutine and a more efficient output routine which prints only specified variables. Forcing-function subroutines to solve implicit equations and generate functions of two variables are planned as additions to the "standard" package.

5. APPENDIX

5.1 Problems in the Evaluation of Exponential Functions

The Taylor series approximation for a scalar exponential function is

$$\epsilon^{y} \approx \sum_{k=0}^{n} \frac{(y)^k}{k!} = 1 + y + \frac{y^2}{2!} + \frac{y^3}{3!} + \ldots + \frac{y^n}{n!} . \qquad (5.1)$$

This approximation also holds true when the argument y is a matrix; hence, matrix exponential functions are amenable to digital computer calculation, since raising a matrix to a power is a straightforward operation.

It is important to note that the HP matrix calculation

$$HP \equiv \left[\exp (A\tau) - I\right]A^{-1} \qquad (5.2)$$

does not require inversion of the A matrix, and can be calculated directly from the terms of the C matrix approximation as shown in Sect. 2.2.

There are several numerical problems associated with the matrix exponential calculations. The approximations will be valid only if

1. the series will converge,
2. the numerical computation does not lose significance due to overflow, roundoff, or truncation errors.

Since the evaluation of exp $(A\tau)$ requires calculating powers of the matrix $A\tau$, there is a practical limitation on the maximum value of the largest element in the $A\tau$ matrix, and experience has shown that it is most efficient to limit this value to about 1.0. Should the desired τ make $\max_{i,j} \left|A_{ij}\tau\right| > 1.0$, then τ is halved up to 10 times for the exponential calculations. The original arguments are restored by applying the following equations as many times as required:

$$C(\tau) \equiv \exp(A\tau) = \exp\left(A\tfrac{\tau}{2}\right)\exp\left(A\tfrac{\tau}{2}\right) \tag{5.3}$$

$$HP(\tau) \equiv \left[\exp(A\tau) - I\right] A^{-1} = \left\{\left[\exp\left(A\tfrac{\tau}{2}\right) - I\right] A^{-1}\right\}\left[I + \exp\left(A\tfrac{\tau}{2}\right)\right] \tag{5.4}$$

There are also provisions in the code to keep track of the roundoff errors in the exponential calculations. The maximum values of the largest elements in the QPT matrices $\frac{(A\tau)^k}{k!}$ are monitored to make sure that they are not larger than the specified precision "P" times 10^8 (for an eight-decimal computer). When the QPT terms are summed, the accuracy of the summation will be approximately P, since the summation is carried out until the largest element in QPT < P. If a maximum value of a QPT element does exceed P x 10^8, then τ is halved, the exponential is calculated, and the original τ is restored as before.

Users are cautioned that roundoff errors may become significant if restoration of the original τ requires very many applications of the argument doubling Eqs. 5.3 and 5.4. We know of no general rules for estimating this limitation; however, checks made on sample problems indicate a "safe" boundary probably exists at a precision P = 10^{-6} and T halved 10 times. With a larger P and more halvings, one should at least be cautious about the results.

The fidelity of the results are also questionable whenever the ratio of the largest (absolute) matrix element to the smallest (nonzero) element is $\geq 10^8$. This might be a manifestation of a very wide range of time constants in a dynamics problem. With a range of $\sim 10^8$, clearly the faster time constants could be considered "instantaneous" with respect to the slower ones, and the equations could probably be rewritten to get around this problem.

5.2 Detailed Description of Programs

Hopefully the information given in this section is sufficient to permit the reader to use and modify MATEXP. Since we have tried going through this typically excruciating experience with programs from others, we have tried making things as clear as possible. In particular, we have used many comment cards in the program listings as a running explanation of what we are doing. Either author would be glad to try to help out any potential MATEXP user, and would be happy to receive any suggestions for improving the program.

5.2.1 MATEXP Main Program

The MATEXP program consists of the main program and two subroutines OUTPUT and DISTRB, plus any other subroutines called by DISTRB. Even if DISTRB is not used, a dummy must be included.

For each case run on MATEXP, the data will include (if appropriate):

1. MATEXP Control Card,
2. Coefficient matrix (A),
3. Initial Condition Vector (XIC),
4. Any data read in by subroutine DISTRB,
5. Fixed forcing function vector (Z).

Input Data Formats - MATEXP Main Program

1. Control Card

Column	1-2		6-7		11-20	21-30	31-40	41-50	51-60	61-62
Format	I2	3X	I2	3X	F10.0	F10.0	F10.0	F10.0	F10.0	I2
Input	NE		LL		P	TZERO	T	TMAX	PLTINC	MATYES

Control Card - cont'd

Column	63-64	65-66	67-69	70	71-72	73-74	75-80
Format	I2	I2	I3	I1	I2	I2	F6.0
Input	ICSS	JFLAG	ITMAX	LASTCC	I1Z	ICONTR	VAR

NE = number of equations

LL = coefficient matrix tag number

P = precision of C and HP - recommend 10^{-6} or less

TZERO = zero time

T = computation time interval

TMAX = maximum time

PLTINC = printing time interval

MATYES = coefficient matrix (A) control flag

1 = use previous A and T

2 = read new coefficients to alter A

3 = read entire new A (nonzero values)

4 = DISTRB to calculate entire new A

5 = read some, DISTRB to calculate others

6 = DISTRB to alter some A elements

ICSS = initial condition vector (XIC) flag

1 = read in all new nonzero values

2 = read new values to alter previous vector

3 = use previous vector

4 = vector = 0

5 = use last value of X vector from previous run

JFLAG = forcing function (Z) flag

1 thru 4 = same as for ICSS for constant Z

5 = call DISTRB at each time step for variable Z

ITMAX = maximum number of terms in series approximation of exp (AT)

LASTCC = nonzero for last case

I1Z = row of Z if only one nonzero, otherwise = 0

ICONTR - for internal control options

0 = read new control card for next case

1 = go to 212 call DISTRB for new A or T

-1 = go to 215 call DISTRB for new initial conditions

VAR = maximum allowable value of largest coefficient matrix element * T
(Recommend VAR = 1.0)

2. Coefficient Matrix A Format 4(2I3, E12.3) - Include if MATYES = 2, 3, or 5.

Column	1-3	4-6	7-18	
Format	I3	I3	E 12.3	Repeat,
Input	Row No.	Col. No.	COEFFICIENT	4 per card

Notes:
1. All row and column number entries on a card must be nonzero.
2. Insert blank card after all coefficient matrix data is read in.
3. Data can be entered in floating point (F) format with decimal point.

3. Initial Condition Vector XIC Format (I2, 5(I3, E12.3))- Include if ICSS = 1 or 2

Column	1-2	3-5	6-17	
Format	I2	I3	E 12.3	Repeat Cols. 3-17,
Input	MM	Row No.	I.C. Value	5 per card

Notes:
1. All row number entries on a card must be nonzero.
2. Insert blank card after all XIC data is read in.
3. Data can be entered in F format.

4. Disturbance Vector Z Format (I2, 5(I3, E12.3))- Include if JFLAG = 1 or 2

Column	1-2	3-5	6-17	
Format	I2	I3	E12.3	Repeat Cols. 3-17,
Input	KK	Row No.	Z Value	5 per card

Note: See notes under 3.

Two figures are included to aid in understanding the MATEXP program. Figure 5.1 summarizes the data arrangement, and Fig. 5.2 is a flow diagram of the main program. The symbols used in MATEXP are also listed and identified.

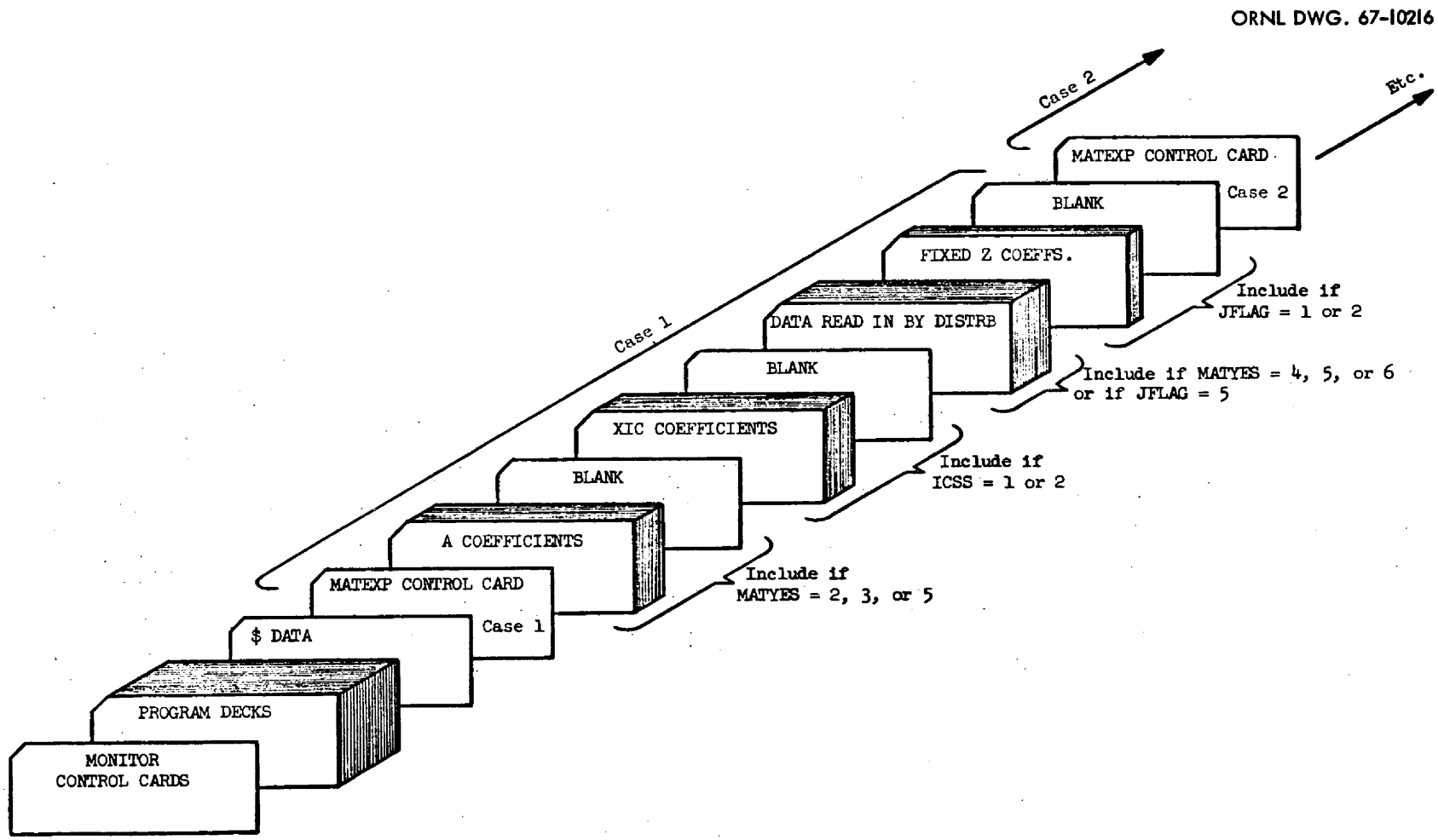


Fig. 5.1 MATEXP Data Arrangement

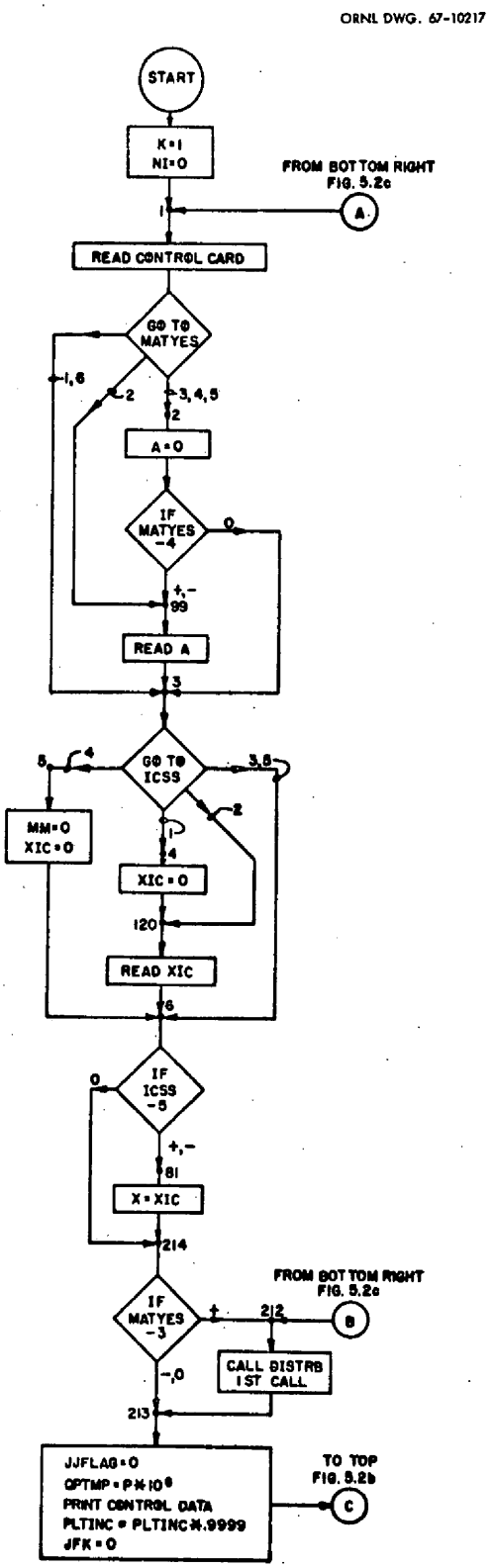


Fig. 5.2a. MATEXP Block Diagram — Read or Compute A Matrix and XIC Vector.

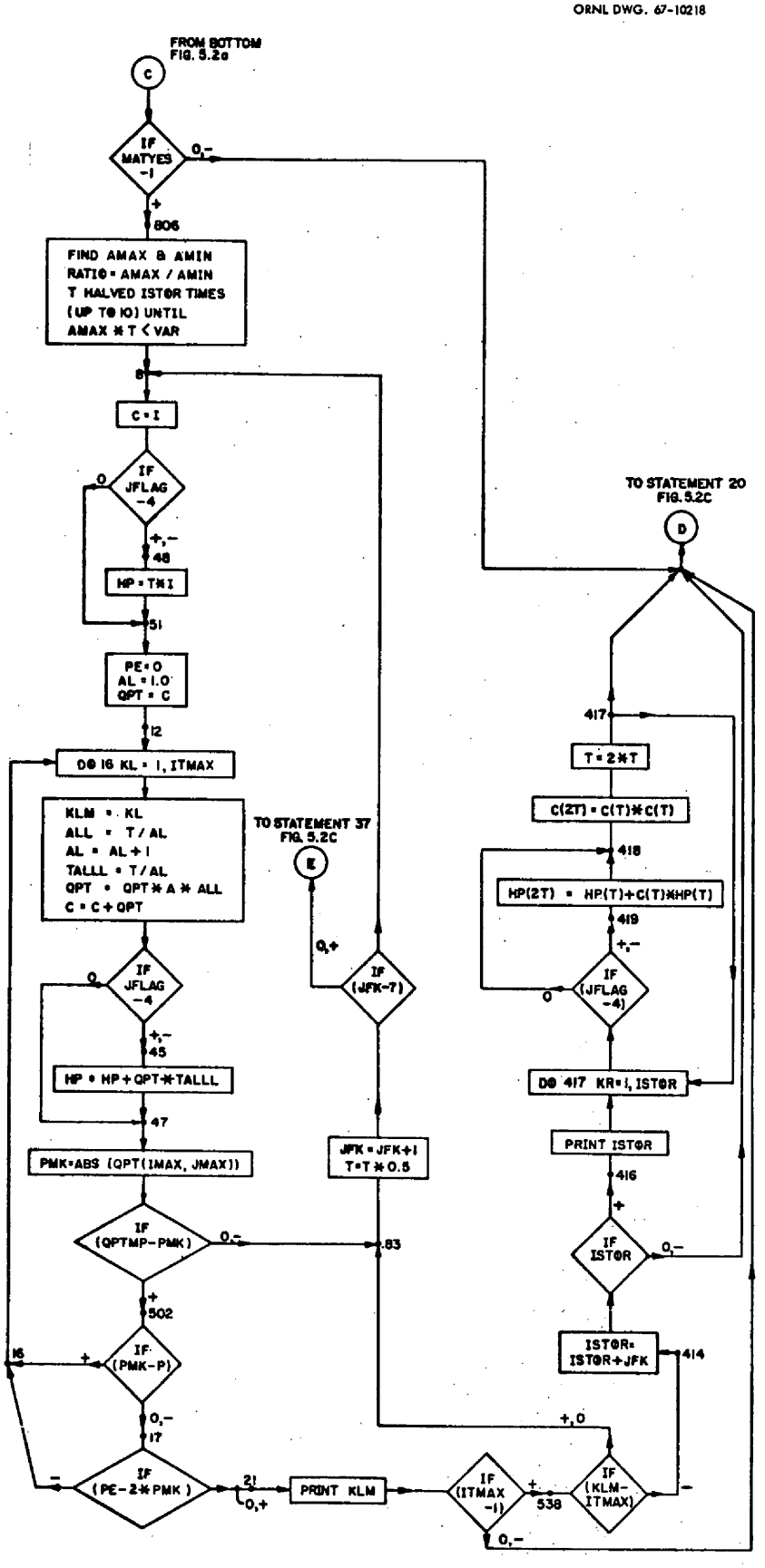


Fig. 5.2b. MATEXP Block Diagram — Compute C and HP Matrices.

ORNL DWG. 67-10219

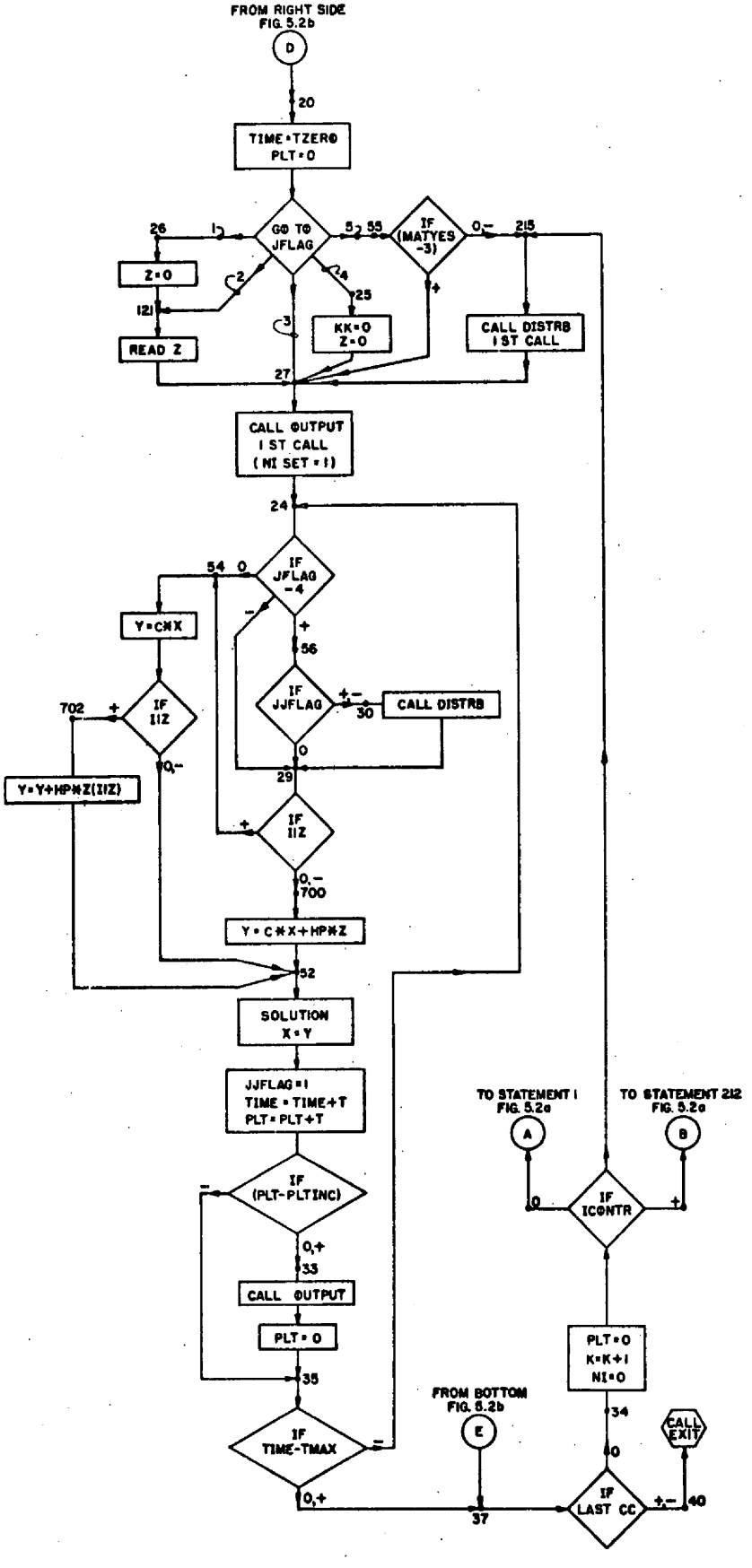


Fig. 5.2c. MATEXP Block Diagram — Compute Solution Vector.

MATEXP MAIN PROGRAM SYMBOL KEY

1. Control Card Inputs

See input data format list.

2. Input Data

A(NE,NE) = coefficient matrix

MM = initial condition vector tag number

XIC (NE) = initial condition vector

KK = disturbance vector tag number

Z(NE) = disturbance vector

3. Internal Variables

The following variables are listed in alphabetical order.

ADT = AMAX * T

AL = Floating point KLM for ALL calc, KLM+1 for TALLL

ALL = T/AL with AL = KLM

AMAX = Maximum (absolute) value of element in A matrix

AMIN = Minimum (absolute) value of nonzero element in A matrix

C(NE,NE) = Coefficient matrix exponential

HP(NE,NE) = Disturbance function matrix exponential

IMAX = Row location of AMAX

IMIN = Row location of AMIN

ISTOR = Number of times matrix exponential argument T is halved so that AMAX * T<VAR; later ISTOR = ISTOR + JFK

JFK = Number of times T is halved in order for matrix exponential calculation precision to be P or better

JJFLAG = Flag to prevent double call of DISTRB during initial time step calculation

JMAX = Column location of AMAX

JMIN = Column location of AMIN

K = Case number

KLM = Number of terms in series approximations of exponentials

NI = Printing flag: 0 on initial call of OUTPUT causing printout of A, C, and HP matrices. OUTPUT sets NI = 1 on first call.

PE = Maximum element in (n - 1)th QPT term

PMK = Maximum element in nth QPT term

QPT(NE,NE) = Term in series approximation of C matrix

QPTMP = Maximum permissible value of element in QPT matrix.

RATIO = AMAX/AMIN. If RATIO less than 10^8 (for eight decimal machine) there may be significant problems in calculation of C and HP.

TALLL = T/AL with AL = KLM +1

TQP(NE) = Temporary storage for QPT terms

X(NE) = Solution vector

Y(NE) = Temporary storage for X

5.2.2 Subroutine OUTPUT

The first time MATEXP calls OUTPUT, the coefficient matrix (A) and the exponential matrices C and HP are printed out, along with the initial solution (X) and disturbance (Z) vectors. OUTPUT also sets the first call flag (NI) to 1, and on subsequent calls only the X and Z vectors are printed. A possible means of saving computing time at the expense of storage would be to store X (and Z) values in arrays for a large number of time intervals, then print the arrays out in blocks. Additional savings could be achieved by printing only selected variables.

5.2.3 Subroutine DISTRB

Subroutine DISTRB may be called by MATEXP either to compute matrix coefficients (A) on the first call (i.e. when flag NI = 0) and/or compute variable forcing-function vectors (Z).

Other special purpose subroutines, such as VARCO, DFG, TRLG, and any others the user may want to supply, are usually called by DISTRB.

Another special purpose use of DISTRB is to compute inputs for successive MATEXP cases without requiring a control card for each case. This is done by means of the flag ICONTR (Cols. 73-4 on the control card). After a case is run, the first call flag NI is reset to 0, and case number K is increased by 1; then if ICONTR is positive, DISTRB will be called at statement 212, where a new

coefficient matrix A or time interval T may be calculated. If ICONTR is negative, DISTRB is called at statement 215, permitting new initial conditions to be used.

The program listing for DISTRB that was used in calculating the sinusoidal forcing function for the example in Sect. 3.3 is given in Sect. 5.3.

Another version of DISTRB is used to calculate the sensitivity of a system's time response to changes in the system's coefficient matrix elements

$$\frac{\partial X}{\partial a_{ij}} .$$

DISTRB controls the solution of the system equations and stores those values of the solution vector which are to be used subsequently as forcing functions for the sensitivity calculations. To compute the sensitivity to a_{ij}, the j^{th} row of the system solution vector is stored and is later used as a forcing function to the i^{th} row of the same system equations.[15]

After solving the system equations and storing the required elements of the response vector, the arithmetic average values of the X's in each time interval are calculated and stored (XT).

During each sensitivity run, DISTRB feeds the forcing function into the system equations, and the resulting printouts of the X vectors are the desired sensitivities.

For the sample program shown in the Fortran listing, Sect. 5.3, the system is forced by a unit step input in row I1Z (specified on the control card). Other control card inputs are:

JFLAG = 5

ICONTR = 1

Special input data read in by DISTRB are the row (IS) and column (JS) numbers of the matrix elements for which sensitivities are to be calculated, the number of time points (NTS), and the number of sensitivity runs (NSENS), as follows:

1			11				51	
IS(1)	JS(1)	(4X)	IS(2)	JS(2)	(4X)	...thru JS(5)	NTI	NSENS
I3	I3		I3	I3			I3	I3

5.2.4 Subroutine VARCO

The VARCO (VARiable COefficient) subroutine can be used with DISTRB to simplify the programming of problems with variable coefficient matrix elements. In general, these elements are functions of both time and the values of the solution vector X. VARCO is designed to be called by DISTRB at the start of each computation interval and to return the mean values of time (TX), and X, (XTR), for that interval. The mean values of X are predicted by a first order extrapolation scheme, as shown in Fig. 3. VARCO will also cause the initial time step to be repeated, using the first try at calculating X(T) to estimate the mean value at $\frac{T}{2}$. DISTRB can then calculate the coefficient values using TX and XTR. Use of this first-order extrapolation scheme results in significant improvement in accuracy over using no extrapolation.

5.2.5 Subroutine DFG

DFG uses the principle of the analog computer's Diode Function Generator (see Fig. 4) and uses linear interpolation to approximate arbitrary, single-valued functions of a variable. Data for DFG is read in the first time it is called by DISTRB (i.e., when NI = 0). The standard program provides for up to 8 functions with up to 32 coordinates each.

On each successive call, DFG returns the functions ZD for varying inputs XD. If an input XD(I) goes outside the specified limits, the output is a straight-line approximation of ZD(I) based on the slope of the function at the boundary, and an error message "DFG(I) RANGE EXCEEDED" is printed.

The inputs read in by DFG are:

NDFGS Number of functions used

NPTS(8) Number of points in approximation for each function

XP(32,8) Independent variable points

ZP(32,8) Dependent variable points

The input format is as follows:

Card No. 1 (I2, 8X, 8I3)

Column	1-2		11-13	Repeat Cols, 11-13 7 more times for NPTS(2) to (7)
Format	I2	8X	I3	
Variable	NDFGS		NPTS(1)	

Card No. 2, 3....etc. (8E10.3)

Column	1-10	11-20	21-30	31-40	Repeat as required for DFG(1); Max. 8 numbers per card
Format	E10.3	E10.3	E10.3	E10.3	
Variable	XP(1,1)	ZP(1,1)	XP(2,1)	ZP(2,1	

NOTES:
1. When all data for DFG(1) has been entered, start DFG(2) data on new card; etc.
2. Enter independent variable points XP in order, progressing from most negative to most positive values.
3. F Format entries (with decimal point) may be used.

5.2.6 Subroutine TRLG

TRLG (TRansport LaG) is described in some detail in Sect. 3.5. The input functions XT (e.g. fluid temperature) and the mass flowrates W (in terms of either mass/time, unity for full flow, or some percentage of full scale) are supplied by the calling program DISTRB, and the lagged functions ZT are returned by TRLG. On the first call of TRLG (when NI = 0), the following input data is read in:

NLAGS Number of functions used

TI(6) Initial value of transport lag time for each function

WMIN(6) Minimum expected value of mass flow W for each function

The program is set up assuming that subroutine VARCO is also called by DISTRB. VARCO has a restart feature which repeats the initial time step calculation; thus the TRLG functions will not be updated on the second call. If VARCO is not used, this second call

omission may be deleted by removing statement 33 in the TRLG program.

The input format for TRLG is:

Card No. 1 (I2)

Column	1-2
Format	I2
Variable	NLAGS

Card No. 2 (6E10.3)

Column	1-10	Repeat 5 more times for TI(2) - (6)
Format	E10.3	
Variable	TI(1)	

Card No. 3 (6E10.3)

Column	1-10	Repeat 5 more times for WMIN(2) - (6)
Format	E10.3	
Variable	WMIN(1)	

5.3 FORTRAN LISTING OF PROGRAMS

```
$IBFTC MAIN    DECK
C      PROGRAM MATEXP  FOR THE 7090 - FORTRAN 4
C
C      THIS PROGRAM CALCULATES THE SOLUTION OF A MATRIX OF FIRST
C      ORDER, SIMULTANEOUS DIFFERENTIAL EQUATIONS W/ CONSTANT COEFFICIENTS
C      OF THE FORM DX/DT # AX + Z.
C
C      THE METHOD IS PAYNTER-S MATRIX EXPONENTIAL METHOD
C
C      THE SOLUTION IS GIVEN FOR INCREMENTS OF THE INDEPENDENT
C      VARIABLE (T) FROM TZERO THROUGH TMAX
C
C      COMPUTES MATRICES C # EXP(A*T) AND
C                        HP # (C-I)*A INVERSE
C      SOLUTION X(N*T) # C*X((N-1)*T)+HP*Z((N-1)*T)
C      SERIES CALCULATION OF C AND HP MONITORED TO
C           ASSURE SPECIFIED SIGNIFICANCE.
C      IF T IS REDUCED FOR C AND HP CALCS.,
C      ORIGINAL ARGUMENTS ARE RESTORED BY -
C           C(2*T)#C(T)*C(T)
C           HP(2*T)#HP(T)+C(T)*HP(T)
C
C      OUTPUT FROM THE PROGRAM IS PRINTED AT INTERVALS PLTINC.
C      THE PROGRAM USES SUBROUTINES DISTRB AND OUTPUT
C
C      INPUT FOR THE PROGRAM CONSISTS OF
C           ONE CONTROL CARD
C           THE COEFFICIENT MATRIX A (UP TO 60 X 60)                    DIM
C           THE INITIAL CONDITION VECTOR X
C           A FIXED DISTURBANCE VECTOR Z
C
C      A VARYING Z CAN BE GENERATED BY DISTRB
C      VARIABLE COEFFICIENT EQUATIONS MAY BE SOLVED BY APPROPRIATE
C      FUDGING OF THE DISTURBANCE FUNCTION SUBROUTINE.
C
C      CONTROL CARD INPUT INFORMATION
C           NE#NO. OF EQUATIONS (I2)
C           LL#COEFF. MATRIX TAG NO.  (I2)
C           P#PRECISION OF C AND HP (F10.0) -  RECOMMEND 1.0E-6 OR LESS
C           TZERO#ZERO TIME  (F10.0)
C           T#COMPUTATION TIME INTERVAL (F10.0)
C           TMAX#MAXIMUM TIME  (F10.0)
C           PLTINC#PRINTING TIME INTERVAL (F10.0)
C           MATYES#COEFF. MATRIX (A) CONTROL FLAG (I2)
C             1#USE PREVIOUS A AND T
C             2#READ NEW COEFF.S TO ALTER A
C             3#READ ENTIRE NEW A  (NON-ZERO VALUES)
C             4#DISTRB TO CALC. ENTIRE NEW A
C             5#READ SOME, DISTRB TO CALC. OTHERS
C             6#DISTRB TO ALTER SOME A ELEMENTS
C           ICSS#INITIAL CONDITION VECTOR (XIC) FLAG (I2)
C             1#READ IN ALL NEW NON-ZERO VALUES
C             2#READ NEW VALUES TO ALTER PREVIOUS VECTOR
C             3#USE PREVIOUS VECTOR
C             4#VECTOR#0
```

```
C             5#USE LAST VALUE OF X VECTOR FROM PREVIOUS RUN
C           JFLAG#FORCING FUNCTION (Z) FLAG (I2)
C             1 THRU 4#SAME AS FOR ICSS FOR CONSTANT Z
C             5#CALL DISTRB AT EACH TIME STEP FOR VARIABLE Z
C      ITMAX # MAX. NO. OF TERMS IN SERIES APPROX.
C           OF EXP(AT).  (I3)
C      LASTCC # NON-ZERO FOR LAST CASE (I1)
C      I1Z # ROW NO. OF Z IF ONLY ONE NON-ZERO,
C           OTHERWISE #0    (I2)
C      ICONTR - FOR INTERNAL CONTROL OPTIONS   (I2)
C           0#READ NEW CONTROL CARD FOR NEXT CASE
C           1#GO TO 212 CALL DISTRB FOR NEW A OR T
C          -1#GO TO 215 CALL DISTRB FOR NEW I.C.-S
C      VAR # MAX. ALLOWABLE VALUE OF LARGEST COEFF. MATRIX ELEMENT * T
C      (RECOMMEND VAR#1.0)   (F6.0)
C
       DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),                    DIMENS
      1X(60),Y(60),Z(60),XIC(60),TQP(60)                                    DIMENS
C
       COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
      1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
      2I1Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
C
C      K#CASE NUMBER
C      NI#0 ON 1-ST PASS.  SET TO 1 ON 1-ST CALL OF OUTPUT.
       K#1
       NI#0
C
    1  READ (5,100)            NE,LL,P,TZERO,T,TMAX,PLTINC,MATYES,ICSS,
      1JFLAG,ITMAX,LASTCC,I1Z,ICONTR,VAR
  100 FORMAT(2(I2,3X),5F10.0,3I2,I3,I1,2I2,F6.0)
C
C      COEFFICIENT MATRIX INPUT
       GO TO (3,99,2,2,2,3),MATYES
    2  DO 90 I#1,NE
       DO 90 J#1,NE
   90  A(I,J)#0.0
       IF(MATYES-4)99,3,99
    99 DO 91 I#1,1379
C      MATRIX ELEMENTS  5(ROW, COLUMN, VALUE)
C      ALL I AND J ENTRIES ON CARD MUST BE NON-ZERO.
C      A BLANK CARD IS REQUIRED AFTER ALL ELEMENTS ARE READ IN.
       READ (5,101)            I1,J1,D1,I2,J2,D2,I3,J3,D3,I4,J4,D4
  101  FORMAT (4(2I3,E12.3))
       IF(I1)3,3,92
    92 A(I1,J1)#D1
       A(I2,J2)#D2
       A(I3,J3)#D3
    91 A(I4,J4)#D4
C
C      INITIAL CONDITION VECTOR XIC INPUT
     3 GO TO(4,120,6,5,6),ICSS
     4 DO 93 I#1,NE
    93 XIC(I)#0.0
   120 DO 94 I#1,15
C      ALL ROW (I) ENTRIES MUST BE NON-ZERO
C      A BLANK CARD IS REQUIRED AFTER ALL ELEMENTS ARE READ IN.
       READ (5,95)            MM,I11,D11,I12,D12,I13,D13,I14,D14,I15,D15
   95  FORMAT(I2,5(I3,E12.3))
```

```
      IF (III)6,6,96
   96 XIC(III)#DII
      XIC(II2)#DI2
      XIC(II3)#DI3
      XIC(II4)#DI4
   94 XIC(II5)#DI5
C
    5 MM#0
      DO 7 I#1,NE
    7 XIC(I)#0.0
    6 IF(ICSS-5)81,214,81
   81 DO 82 I#1,NE
   82 X(I)#XIC(I)
  214 IF(MATYES-3)213,213,212
  212 CALL DISTRB
  213 JJFLAG#0
C     QPTMP # MAX. PERMISSIBLE ELEMENT OF QPT FOR  8 DECIMAL COMPUTER
C     MATRIX CALC. LOSES SIGNIFICANCE IF LARGEST
C       ELEMENT IN SERIES APPROX. MATRIX QPT IS
C       GREATER THAN P*1.0E8
      QPTMP#P*1.0E8
C
      WRITE (6,211)            K,NE,P,T,
     1PLTINC,MATYES,ICSS,JFLAG,ICONTR,ITMAX,IIZ,VAR,QPTMP
C
  211 0FORMAT(12H1MATEXP CASE,I3/17H NO. OF EQUATIONS,
     1I3/20H SPECIFIED PRECISION,F12.8/6H TIME ,
     28HINTERVAL,F18.8/15H PLOT INCREMENT,F17.8//
     316H CONTROL FLAGS -/1H ,5X,6HMATYES,I4/1H ,
     45X,4HICSS,I6/1H ,5X,5HJFLAG,I5/1H ,5X,6HICONTR,I4/
     534H0MAX. TERMS IN EXPONENTIAL APPROX.,I5/
     613H SINGLE Z ROW,I4/20H MAX. ALLOWABLE A*DT,F9.3/
     727H MAX. ALLOWABLE QPT ELEMENT,F11.3)
C
      PLTINC#PLTINC*0.9999
C
      JFK#0
      IF(MATYES-1)20,20,806
C     SCAN MATRIX FOR MAX. AND MIN. NON-ZERO ELEMENTS.
  806 IMAX#1
      JMAX#1
      AMAX#ABS (A(1,1))
      DO 401 I#1,NE
      DO 401 J#1,NE
      IF(AMAX-ABS (A(I,J)))402,401,401
  402 AMAX#ABS (A(I,J))
      IMAX#I
      JMAX#J
  401 CONTINUE
      IMIN#IMAX
      JMIN#JMAX
      AMIN#AMAX
      DO 409 I#1,NE
      DO 409 J#1,NE
      IF(A(I,J)) 407,409,407
  407 IF(ABS (A(I,J))-AMIN) 408,409,409
  408 AMIN#ABS (A(I,J))
      IMIN#I
      JMIN#J
```

```
 409  CONTINUE
      RATIO#AMAX/AMIN
C     AMIN # MINIMUM NON-ZERO ELEMENT
      ISTOR#0
      ADT#AMAX*T
      DO 403 I#1,11
      IF(VAR-ADT) 413,404,404
  413 ISTOR#ISTOR+1
 403  ADT#ADT*0.5
 404  T#ADT/AMAX
C     COMPUTATION INTERVAL T IS HALVED ISTOR
C       TIMES (10#MAX.) SO MAX. ELEMENT IN A*T
C       IS LESS THAN VAR.
      WRITE (6,405)            IMAX,JMAX,A(IMAX,JMAX),ADT,T,
     1 IMIN,JMIN,A(IMIN,JMIN),RATIO
 405  FORMAT (31H0MAX.COEFF. MATRIX ELEMENT # A(,I2,1H,,I2,3H) #,
     1 E15.4/13H MAX. A*DT # ,F12.8,2X,14HWITH DELTA T #,F15.8/
     230H0MINIMUM NON-ZERO ELEMENT # A(,I2,1H,,I2,3H) #,E15.4/
     318H RATIO AMAX/AMIN #,E15.4)
C
      IF(ISTOR-10)8,410,410
  410 WRITE (6,411)
  411 0FORMAT(34H0A*DT STILL GREATER THAN ALLOWABLE,
     119H AFTER 10 HALVINGS.)
      GO TO 37
C     CALCULATION OF MATRIX EXPONENTIALS C AND HP
    8 DO 9 I#1,NE
      DO 9 J#1,NE
    9 C(I,J)#0.
C
      DO 10 I#1,NE
   10 C(I,I)#1.
C
C     SKIP HP CALCS. FOR HOMOGENEOUS EQUATIONS
      IF (JFLAG-4)48,51,48
   48 DO 49 I#1,NE
      DO 49 J#1,NE
   49 HP(I,J)#0.
C
      DO 50 I#1,NE
   50 HP(I,I)#T
C
   51 PE#0.0
C
      DO 11 I#1,NE
      DO 11 J#1,NE
   11 QPT(I,J)#C(I,J)
C
C NOW FORM THE MATRIX EXPONENTIALS C#EXP(A*T) AND HP#((C-I)*A INVERSE)
C
      AL#1.0
C
   12 DO 16 KL#1,ITMAX
C
      KLM#KL
      ALL#T/AL
      AL#AL+1.0
      TALL#T/AL
C
```

```
      DO 18 I#1,NE
C
C
      DO 13 J#1,NE
      TQP(J)#0.0
      DO 13 KX#1,NE
   13 TQP(J)#TQP(J)+QPT(I,KX)*A(KX,J)
C
      DO 18 J#1,NE
   18 QPT(I,J)#TQP(J)*ALL
C
C     QPT#MATRIX TERM IN SERIES APPROX. #((A*T)**K)/K FACTORIAL
C
      DO 44 I#1,NE
      DO 44 J#1,NE
   44 C(I,J)#C(I,J)+QPT(I,J)
C
      IF (JFLAG-4)45,47,45
C
   45 IF(ITMAX-KL)47,47,145
  145 DO 46 I#1,NE
      DO 46 J#1,NE
   46 HP(I,J)#HP(I,J)+QPT(I,J)*TALLL
C
C
C FIND MAX ABS ELEMENT IN QPT AND CALL IT PMK
C
C     LARGEST QPT ELEMENT USUALLY IN ROW IMAX, COLUMN JMAX
   47 PMK#ABS (QPT(IMAX,JMAX))
      IF(QPTMP-PMK) 83,83,502
  502 IF(PMK-P) 406,406,16
C     SCAN OTHER QPT ELEMENTS ONLY WHEN QPT(IMAX, JMAX) IS LESS THAN P
  406 DO 14 I#1,NE
      DO 14 J#1,NE
   14 PMK#AMAX1(PMK,ABS (QPT(I,J)))
      IF(PMK-P)17,17,16
C
C     PRESENT MAX. QPT ELEMENT SHOULD BE LESS THAN
C       HALF PREVIOUS MAX. TO INSURE CONVERGENCE
   17 IF(PE-2.*PMK)16,21,21
   16 PE#PMK
C
   21 WRITE (6,200)          KLM
C
  200 FORMAT(44H0NO. OF TERMS IN SERIES APPROX. OF MATEXP # ,I2)
C
      IF(ITMAX-1)20,20,538
  538 IF(KLM-ITMAX) 414,83,83
C
   83 T#T*0.5
      JFK#JFK+1
      IF(JFK-7)303,304,304
  304 WRITE (6,305)          PMK
  305 0FORMAT(32H07 TRIES AT HALVING T N.G., PMK#,F12.6)
      GO TO 37
  303 WRITE (6,210)          KLM,PMK,T
  210 FORMAT(21H0MAX. ELEMENT IN TERM,I3,8HOF QPT #,E11.3/
     1 35H TRY HALVED TIME INTERVAL DELTA T #,F15.8)
      GO TO 8
```

```
  414 ISTOR#ISTOR+JFK
C     ORIGINAL ARGUMENTS OF C AND HP MATRICES RESTORED IF ISTOR GREATER THAN 0
      IF(ISTOR) 20,20,416
  416 WRITE (6,415)            ISTOR
  415 FORMAT(26H0TOTAL NO. OF T HALVINGS #,I3)
      DO 417 KR#1,ISTOR
      IF(JFLAG-4) 419,418,419
C     SKIP HP CALCS. FOR HOMOGENEOUS EQUATIONS
  419 DO 420 I#1,NE
      DO 421 J#1,NE
      TQP(J)#0.0
      DO 421 KX#1,NE
  421 TQP(J)#TQP(J)+HP(I,KX)*C(KX,J)
      DO 420 J#1,NE
  420 HP(I,J)#TQP(J)+HP(I,J)
C
  418 DO 430 I#1,NE
      DO 430 J#1,NE
  430 QPT(I,J)#0.0
      DO 431 I#1,NE
      DO 431 J#1,NE
      DO 431 KX#1,NE
  431 QPT(I,J)#QPT(I,J)+C(I,KX)*C(KX,J)
      DO 432 I#1,NE
      DO 432 J#1,NE
  432 C(I,J)#QPT(I,J)
  417 T#2.0*T
C
C     C(I,J) IS THE MATRIX EXPONENTIAL C#EXP(A*T)
C     AND HP(I,J) IS THE ((C-I)*A INVERSE) MATRIX
C NOW WE READ (OR CALL SUBROUTINE FOR) DISTURBANCE VECTOR
C
   20 TIME#TZERO
      PLT#0.
      GO TO (26,121,27,25,55),JFLAG
   55 IF(MATYES-3)215,215,27
  215 CALL DISTRB
      IIZ#IIZ
      GO TO 27
C
   26 DO 97 I#1,NE
   97 Z(I)#0.0
  121 DO 98 I#1,15
C     ALL ROW (I) ENTRIES MUST BE NON-ZERO
C     A BLANK CARD IS REQUIRED AFTER ALL ELEMENTS ARE READ IN.
      READ (5,95)            KK,I21,D21,I22,D22,I23,D23,I24,D24,I25,D25
      IF(I21)27,27,78
   78 Z(I21)#D21
      Z(I22)#D22
      Z(I23)#D23
      Z(I24)#D24
   98 Z(I25)#D25
C
   25 KK#0
      DO 28 I#1,NE
   28 Z(I)#0.
C
C     ON 1-ST CALL OF OUTPUT NI SET TO 1
   27 CALL OUTPUT
```

```
C
C NOW COMES THE EQUATION SOLUTION BASED ON
C      X(NT)#M*X(NT-1)+((M-1)A INV.)*Z(NT-1)
C
   24 IF (JFLAG-4)29,54,56
   54 DO 53 I#1,NE
      Y(I)#C(I,1)*X(1)
      DO 53 J#2,NE
   53 Y(I)#Y(I)+C(I,J)*X(J)
      IF(IIZ)52,52,702
   56 IF(JJFLAG)30,29,30
   30 CALL DISTRB
   29 IF(IIZ)700,700,54
C     ONLY ONE Z-TERM CALC. IF IIZ IS GREATER THAN ZERO
  702 DO 703 I#1,NE
  703 Y(I)#Y(I)+HP(I,IIZ)*Z(IIZ)
      GO TO 52
  700 DO 32 I#1,NE
      Y(I)#C(I,1)*X(1)+HP(I,1)*Z(1)
      DO 32 J#2,NE
   32 Y(I)#Y(I)+C(I,J)*X(J)+HP(I,J)*Z(J)
   52 DO 31 I#1,NE
   31 X(I)#Y(I)
C
C ONE TIME INCREMENT OF THE SOLUTION HAS JUST BEEN FOUND
C NOW PLOT AND PRINT IF PLTINC INTERVAL HAS ELAPSED
C
      JJFLAG#1
      TIME#TIME+T
      PLT#PLT+T
      IF(PLT-PLTINC)35,33,33
   33 CALL OUTPUT
      PLT#0.
   35 IF(TIME-TMAX)24,37,37
   37 IF(LASTCC)40,34,40
   34 K#K+1
      NI#0
      PLT#0.0
      IF(ICONTR)215,1,212
   40 STOP
      END
```

```
$IBFTC OUT      DECK
      SUBROUTINE OUTPUT
C
C
      DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),                    DIMENS
     1X(60),Y(60),Z(60),XIC(60),TQP(60)                                    DIMENS
C
      COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
     1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
     2I1Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
C
      IF(NI)2,1,2
    1 NI#1
      NC#10
      DO 11 NCM#1,51,10
      WRITE(6,200) LL,((A(I,J),J#NCM,NC),I#1,NE)
  200 FORMAT (2H0A,I2/(1H ,1P10E11.3))
      IF(NE-NC) 10,10,11
   11 NC#NC+10
C
   10 NC#10
      DO 21 NCM#1,51,10
      WRITE(6,201) ((C(I,J),J#NCM,NC),I#1,NE)
  201 FORMAT (2H0C/(1H ,1P10E11.3))
      IF(NE-NC) 20,20,21
   21 NC#NC+10
C
   20 NC#10
      DO 31 NCM#1,51,10
      WRITE(6,202) ((HP(I,J),J#NCM,NC),I#1,NE)
  202 FORMAT (3H0HP/(1H ,1P10E11.3))
      IF(NE-NC) 2,2,31
   31 NC#NC+10
C
    2 WRITE(6,203) TIME,(X(I),I#1,NE)
  203  FORMAT(4H T #,1PE10.3,4H X #,       /(1H ,5X,10E11.3))
      IF(JFLAG.NE.5) GO TO 30
C
      WRITE(6,204) (Z(I),I#1,NE)
  204  FORMAT(6H0Z #  ,1P10E11.3/(1H ,5X,10E11.3))
   30  RETURN
      END
```

```
$IBFTC SUBZ     DECK
       SUBROUTINE DISTRB
C
C      DISTRB FOR REPORT EXAMPLE
C
       DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),                    DIMENS
      1X(60),Y(60),Z(60),XIC(60),TQP(60)                                    DIMENS
       COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
      1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
      2II,Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
C
       TX#TIME+0.5*T
       Z(I)#SIN (2.0*TX)
       RETURN
       END



$IBFTC DSENS    DECK
       SUBROUTINE DISTRB
C      DISTRB FOR TIME RESPONSE SENSITIVITY CALCS.
       DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),
      1X(60),Y(60),Z(60),XIC(60),TQP(60)
       COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
      1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
      2II,Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
       DIMENSION IR(5),IS(15),JS(15),IQ(30),XT(5,1000),
      1XSEN(15,30),XPSI(30)
       IF(NI)1,1,2                                                          29880105
     1 IF(ICONTR+2)5,4,3                                                    29880107
     2 IF(ICONTR+2)7,6,6                                                    29880108
C      INITIAL INPUTS AND CALCS.
   3   READ(5,100)(IS(I),JS(I),I#1,5),NTI,NSENS
   100 FORMAT(6(2I3,4X))
       NDT#1                                                                29880113
       ICONTR#-2                                                            29880115
       NTIMO#NTI-1                                                          29880117
       DO 8 I#1,NE
   8   Z(I)#0.0
C      DURING SOLUTION OF SYSTEM EQUATIONS
     6 DO 20 I#1,NSENS
       ICO#JS(I)
    20 XT(I,NDT)#X(ICO)                                                     29880123
       NDT#NDT+1
       GO TO 30                                                             29880201
C                                                                           29880212
C      JUST AFTER SYSTEM SOLUTION IS COMPLETED
     4 IST#0                                                                29880203
       ICONTR#-3                                                            29880205
       DO 21 I#1,NSENS
       DO 21 J#1,NTIMO                                                      29880209
    21 XT(I,J)#0.5*(XT(I,J)+XT(I,J+1))                                      29880211
C      XT # AVG VALUES OF SENSITIVITY EQN INPUTS                            29880213
       WRITE(6,102)            ((XT(I,J),J#1,NTI),I#1,NSENS)
   102 FORMAT(3H0XT/(1H ,10E11.3))
C                                                                           29880214
C      AFTER COMPLETING EACH SENSITIVITY RUN -
     5 IST#IST+1                                                            29880215
       IF(IST-NSENS)31,31,32                                                29880217
```

```
C       GO TO NEXT CASE
   32   ICONTR#0
        PLTINC#TMAX
        TMAX#0.0
        NI#1
        GO TO 30
    31  IIZ#IS(IST)                                                    29880219
C       COL. IIZ OF HP MATRIX MULT. BY Z                               29880221
        WRITE(6,101)              IS(IST),JS(IST)
   101  FORMAT(18H0SENSITIVITY TO A(,I3,1H,,I3,1H))                    29880301
        TIME#TZERO                                                     29880303
        NDT#1
        DO 41 I#1,NE                                                   29880305
        X(I)#0.0
    41  Z(I)#0.0                                                       29880309
        JJFLAG#0
C       DURING EACH SENSITIVITY RUN -
     7  Z(IIZ)#XT(IST,NDT)
        NDT#NDT+1
    30  RETURN                                                         29880315
        END                                                            29880317
```

```
$IBFTC SUBV    DECK
      SUBROUTINE VARCO(XTR,TX)                                          29880101
C     FOR USE WITH DISTRB AND MATEXP FOR                                29880103
C     VARIABLE Z-S.  GIVES 1-ST ORDER EXTRAP.                           29880105
C     FOR AVG. X AND TIME, PLUS RESTART                                 29880107
C     ON 1-ST INTERVAL.  DISTRB FORM #                                  29880109
C          CALC. MATRIX COEFF.-S, ETC. IF NI#0                          29880111
C          CALL VARCO(XTR,TX)                                           29880113
C          CALC. Z-S USING XTR(I)-S AND TX (TIME).                      29880115
C                                                                       29880117
      DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),                 DIMENS
     1X(60),Y(60),Z(60),XIC(60),TQP(60)                                 DIMENS
      COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
     1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
     2I1Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
      DIMENSION XTR(60),XL(60)                                          29880118
C
      IF(NI)1,1,2                                                       29880120
C     FIRST ENTRY                                                       29880121
    1 NV#1                                                              29880122
      TX#TZERO+0.5*T                                                    29880124
      DO 10 I#1,NE                                                      29880202
   10 XTR(I)#XIC(I)                                                     29880204
      GO TO 30                                                          29880206
    2 IF(NV)3,3,4                                                       29880208
C     SECOND ENTRY                                                      29880210
    4 NV#0                                                              29880212
      TIME#TZERO                                                        29880214
      PLT#0.0
      DO 11 I#1,NE                                                      29880216
      XL(I)#XIC(I)                                                      29880218
      XTR(I)#0.5*(XL(I)+X(I))                                           29880220
   11 X(I)#XIC(I)                                                       29880222
      GO TO 30                                                          29880224
C     ENTRIES AFTER SECOND                                              29880301
    3 TX#TIME+0.5*T                                                     29880303
      DO 12 I#1,NE                                                      29880305
      XTR(I)#X(I)+0.5*(X(I)-XL(I))                                      29880307
   12 XL(I)#X(I)                                                        29880309
   30 RETURN                                                            29880311
      END                                                               29880313
```

```
$IBFTC FGEN    DECK
      SUBROUTINE DFG(XD,ZD)
C                                                                      29880105
C     EQUIVALENT TO 8 DFG-S WITH UP TO 32                              29880106
C     POINTS EACH.  CALLED BY DISTRB.                                  29880107
C                                                                      29880108
C     INPUTS ARE                                                       29880109
C       NDFGS  NO. OF DFG-S USED                                       29880112
C       NPTS  NO. OF POINTS IN EACH DFG                                29880113
C       XP  INDEPENDENT VARIABLE DFG POINTS                            29880110
C       ZP  DEPENDENT VARIABLE DFG POINTS
C                                                                      29880114
C     XD IS THE INPUT VARIABLE AND ZD THE OUTPUT                       29880115
C                                                                      29880116
      DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),                DIMENS
     1X(60),Y(60),Z(60),XIC(60),TQP(60)                                DIMENS
      COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
     1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
     2I1Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
      DIMENSION XP(32,8),ZP(32,8),SL(32,8),NPTS(8),                    29880117
     1JP(8),ZD(8),XD(8)                                                29880118
C                                                                      29880119
C                                                                      29880121
      IF(NI)1,2,1                                                      29880122
C     FIRST CALL COMP.                                                 29880123
C                                                                      29880124
    2 READ (5,100)            NDFGS,NPTS
  100 FORMAT(I2,8X,8I3)
      DO 86 I#1,NDFGS
      NP#NPTS(I)
    7 READ (5,101)            (XP(J,I),ZP(J,I),J#1,NP)
  101 FORMAT(8E10.3)
   86 WRITE (6,200)            I,(XP(J,I),ZP(J,I),J#1,NP)
 2000FORMAT(4H0DFG,I3,17H XP AND ZP INPUTS/
     1(1H0,4(2E12.4,4X)))
      DO 3 I#1,NDFGS                                                   29880125
      M#NPTS(I)-1                                                      29880201
      DO 3 J#1,M                                                       29880202
    3 SL(J,I)#(ZP(J+1,I)-ZP(J,I))/(XP(J+1,I)-XP(J,I))
C                                                                      29880204
      DO 5 I#1,NDFGS                                                   29880205
      DO 4 J#2,32
      IF(XD(I)-XP(J,I))5,5,4                                           29880207
    4 CONTINUE                                                         29880208
    5 JP(I)#J                                                          29880209
C                                                                      29880210
C     CALCS. MADE EACH TIME                                            29880211
    1 DO 6 I#1,NDFGS                                                   29880212
      J#JP(I)                                                          29880213
   18 IF(XD(I)-XP(J,I))10,11,12                                        29880214
   10 IF(XD(I)-XP(J-1,I))13,14,15                                      29880215
   13 J#J-1                                                            29880216
      IF(J-1)16,16,10
   16 J#2
      GO TO 19
   14 ZD(I)#ZP(J-1,I)                                                  29880218
      GO TO 6                                                          29880219
   12 J#J+1                                                            29880220
      IF(NPTS(I)-J)17,18,18
```

```
   17 J#NPTS(I)
      GO TO 19
   11 ZD(I)#ZP(J,I)                                                  29880222
      GO TO 6                                                        29880223
   19 WRITE (6,102) I
  102 FORMAT(4H□DFG,I3,16H RANGE EXCEEDED.)
C                                                                    29880224
   15 ZD(I)#ZP(J-1,I)+SL(J-1,I)*(XD(I)-XP(J-1,I))                    29880225
C     JP(I) STORES VALUE OF XD LOCATION
C       TO USE AS FIRST TRY NEXT TIME.
    6 JP(I)#J                                                        29880301
C                                                                    29880302
      RETURN                                                         29880303
      END                                                            29880304
```

```
$IBFTC TRLAG   DECK
      SUBROUTINE TRLG(XT,W,ZT)
C
C     VARIABLE TRANSPORT LAG GENERATOR -  FORTRAN IV
C
C     USES UP TO 300 POINT APPROXIMATION FOR                          29880105
C     UP TO 6 VARIABLES.  USES INVENTORY CALC.                        29880106
C
C     INPUTS FOR EACH LAG (TOTAL # NLAGS)                             29880108
C       1. INPUT FUNCTION XT(I)                                       29880109
C       2. MASS FLOWRATE W(I)                                         29880110
C       3. INITIAL VALUE OF LAG TIME TI(I)                            29880111
C       4. MINIMUM EXPECTED VALUE OF MASS FLOW WMIN(I)                29880112
C
C     OUTPUTS ARE LAGGED FUNCTIONS ZT(I)                              29880114
C
      DIMENSION A(60,60),C(60,60),HP(60,60),QPT(60,60),               DIMENS
     1X(60),Y(60),Z(60),XIC(60),TQP(60)                               DIMENS
      COMMON C,HP,A,QPT,X,Z,Y,ITMAX,KK,LL,MM,
     1JJFLAG,XIC,NI,TIME,TMAX,TZERO,NE,TQP,T,
     2II,Z,ICONTR,PLTINC,MATYES,ICSS,JFLAG,PLT
      DIMENSION XT(6),W(6),TI(6),WMIN(6),ZT(6),XS(300,6),             DIMENS
     1PS(300,6),KT(6),JT(6),XJMP(6),JMP(6),NJMP(6)                    DIMENS
C
C     NI # 1-ST CALL FLAG  (# 0 ON 1-ST CALL)
C     T # COMPUTATION TIME INTERVAL
C
      IF(NI)20,21,20                                                  29880121
C     FIRST CALL COMP.                                                29880123
   21 READ(5,100) NLAGS,TI,WMIN
  100 FORMAT(I2/(6E10.3))                                             DIMENS
      WRITE(6,101) TI,WMIN
  101 FORMAT(26H0TRLG INPUTS - TI AND WMIN/(1H0,6E18.5))              DIMENS
      DO 22 I#1,NLAGS
      XJMP(I)#1.0                                                     29880202
      XS(1,I)#XT(I)                                                   29880203
      PS(1,I)#W(I)*TI(I)                                              29880204
      XNSP#PS(1,I)/(WMIN(I)*T)
      DO 23 M#1,10                                                    29880206
      P1#XJMP(I)*XNSP
      IF(300.0-P1)23,24,24                                            DIMENS
   23 XJMP(I)#XJMP(I)+1.0                                             29880209
C
   24 JMP(I)#IFIX(XJMP(I))
      KT(I)#2                                                         29880212
      JT(I)#1                                                         29880213
   22 NJMP(I)#1                                                       29880214
      NV#-1
C
C     CALCS. MADE EACH TIME                                           29880216
   20 NV#NV+1
C ****** NOTE - IF A RESTART FEATURE IS USED (WHERE THE INITIAL TIME
C     STEP CALCULATION IS REPEATED), THE FLAG NV AND STATEMENT 33 WILL
C     OMIT THE TRLG CALC.  THIS 1-ST CALL OMISSION MAY BE DELETED BY
C     REMOVING STATEMENT 33.
   33 IF(NV)31,32,31
   31 DO 17 I#1,NLAGS
      IF(NJMP(I)-JMP(I))26,27,27                                      29880218
   26 NJMP(I)#NJMP(I)+1                                               29880219
```

```
      GO TO 17                                                          29880220
   27 NJMP(I)#1                                                         29880221
      K#KT(I)                                                           29880222
      J#JT(I)                                                           29880223
      XS(K,I)#XT(I)                                                     29880224
      PS(K,I)#XJMP(I)*W(I)*T
C     J#NO. OF ELEMENT AT EXIT.  K#NO. AT ENTRANCE                      29880301
      IF(PS(J,I)-PS(K,I))1,2,3                                          29880302
    2 ZT(I)#XS(J,I)                                                     29880303
      IF(J-300)6,7,7                                                    DIMENS
    7 JT(I)#1                                                           29880305
      GO TO 30                                                          29880306
    6 JT(I)#J+1                                                         29880307
      GO TO 30                                                          29880308
C                                                                       29880309
    1 COLLT#XS(J,I)                                                     29880310
      COLLP#PS(J,I)                                                     29880311
      DO 15M#1,300                                                      DIMENS
      IF(J-300)8,9,9                                                    DIMENS
    9 J#0
    8 J#J+1                                                             29880316
      PQ#COLLP+PS(J,I)
C
      IF(PQ-PS(K,I))11,12,13                                            29880319
   11 COLLT#(COLLT*COLLP+XS(J,I)*PS(J,I))/PQ                            29880320
C
   15 COLLP#COLLP+PS(J,I)
C
   12 ZT(I)#(COLLT*COLLP+XS(J,I)*PS(J,I))/PQ                            DIMENS
C
      IF(J-300)14,16,16                                                 29880401
   16 JT(I)#1                                                           29880402
      GO TO 30                                                          29880403
   14 JT(I)#J+1                                                         29880404
      GO TO 30                                                          29880405
C
   13 PS(J,I)#PQ-PS(K,I)                                                29880407
      ZT(I)#(COLLT*COLLP+XS(J,I)*PS(J,I))/(COLLP+PS(J,I))               29880408
      JT(I)#J                                                           29880409
      GO TO 30                                                          29880410
C
    3 ZT(I)#XS(J,I)                                                     29880412
      PS(J,I)#PS(J,I)-PS(K,I)                                           29880413
C
   30 IF(K-300)4,5,5                                                    DIMENS
    5 KT(I)#1                                                           29880416
      GO TO 17                                                          29880417
    4 KT(I)#K+1                                                         29880418
   17 CONTINUE                                                          29880419
C
   32 RETURN
      END
```

ORNL-TM-1933

INTERNAL DISTRIBUTION

1-30. R. K. Adams
31. L. L. Anthony (K-25)
32-61. S. J. Ball
62. T. V. Blalock
63. C. J. Borkowski
64. J. B. Bullock
65. O. W. Burke
66. F. H. Clark
67. R. A. Dandl
68. H. P. Danforth
69. J. B. Davidson
70. S. J. Ditto
71. B. C. Duggins
72. J. R. Engel
73. E. P. Epler
74. G. J. Farris
75. A. H. Fowler (Y-12)
76. T. B. Fowler
77. D. N. Fry
78. E. W. Hagen
79. C. S. Harrill
80. R. F. Hyland
81. G. R. Jamieson
82. W. H. Jordan
83. S. V. Kaye
84. R. J. Kedl
85. T. S. Kress
86. R. C. Kryter
87. C. G. Lawson
88. J. L. Lucius
89. J. B. Mankin, Jr. (K-25)
90. C. D. Martin
91. W. J. McClain (K-25)
92. H. A. McLain
93. H. J. Metz
94. R. V. Miskell (Y-12)
95. R. L. Moore
96. C. A. Mossman
97. C. W. Nestor
98. C. H. Nowlin
99. H. G. O'Brien
100. J. S. Olsen
101. C. L. Partain
102. B. C. Patten
103. R. W. Peele
104. A. M. Perry
105. P. H. Pitkanen
106. B. E. Prince
107. D. P. Roux
108. G. S. Sadowski
109. W. K. Sartory
110. A. N. Smith
111. O. L. Smith
112. B. Squires
113. R. S. Stone
114. J. R. Tallackson
115. M. L. Tobias
116. J. R. Trinko
117. T. C. Tucker
118. C. S. Walker
119. J. S. Watson
120. M. E. Whatley
121. R. E. Whitt
122. R. P. Wichner
123. J. V. Wilson
124. M. M. Yarosh
125-126. Central Research Library
127. Document Reference Section
128-132. Laboratory Records Department
133. Laboratory Records, ORNL R.C.
134. ORNL Patent Office
135-149. Division of Technical Information Ext.
150. Research and Development Division ORO

EXTERNAL DISTRIBUTION

151. S. G. Bloom, Battelle Memorial Institute, Columbus, Ohio
152. Neal Carter, Battelle Northwest Labs., Richland, Washington
153. E. H. Cooke-Yarborough, Electronics Division, AERE, Harwell, England
154. S. J. Gage, University of Texas
155. R. P. Gardner, North Carolina State University
156. B. E. Gilliland, Clemson University
157. S. H. Hanauer, University of Tennessee
158. T. H. M. Hung, Deere & Company, Moline, Illinois
159. J. M. Jansen, Jr., University of Illinois
160. T. W. Kerlin, University of Tennessee
161. L. B. Koppel, Purdue University
162. J. B. H. Kuper, Brookhaven National Laboratory
163. Rufus Oldenburger, Purdue University
164. H. M. Paynter, Massachusetts Institute of Technology
165. J. W. Prados, University of Tennessee
166. J. C. Robinson, University of Tennessee
167. R. F. Saxe, North Carolina State University
168. E. Siddall, AECL, Chalk River, Ontario
169. S. E. Stephenson, University of Arkansas
170. Otis Updike, University of Virginia
171. T. J. Williams, Purdue University
172. W. C. Wright, University of Tennessee