UNCLASSIFIED

CLM-PDN 3/73

A PROGRAM FOR THE SOLUTION OF BOUNDARY VALUE PROBLEMS FOR SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS

by

R. ENGLAND*

ABSTRACT

This report constitutes the documentation of a program which has been developed for the solution of two point boundary value and eigenvalue problems in differential equations. Because of the multiple shooting technique used it is capable of solving a great variety of such problems. The program may be used on its own, or as a subroutine within a larger program, using either its standard input and output, or communicating via argument lists. Three sample problems are described, from the areas of hydrodynamics, atomic physics and plasma physics. The difficulties encountered are described, and full details are provided of the working of the program.

* Presently at: Universidad Autónoma Metropolitana,
Unidad Iztapalapa,
Departamento de Matemáticas,
Apdo. Postal 55 - 534,
México 13.D.F.

Correspondence should be addressed to: Roland England,
Antonio Sola 45,
Col. Condesa,
México 11.D.F.

U.K.A.E.A. Research Group,
Culham Laboratory,
Abingdon,
Oxon.

October, 1975

CONTENTS

1. PURPOSE OF THE PROGRAM

This report describes a program which is designed to solve numerically a system of Ordinary Differential Equations constituting a Two Point Boundary Value or Eigenvalue Problem.

A common form of two point boundary value problem may be written:

$$\frac{d^2y}{dt^2} + \varphi\left(t, y, \frac{dy}{dt}\right)\frac{dy}{dt} + \chi\left(t, y, \frac{dy}{dt}\right)y = \psi\left(t, y, \frac{dy}{dt}\right) , \tag{1}$$

$$\alpha_1 \frac{dy}{dt}(a) + \beta_1\, y(a) = \gamma_1 , \qquad \alpha_2 \frac{dy}{dt}(b) + \beta_2\, y(b) = \gamma_2 . \tag{2}$$

Before submission to this program, such a problem would have to be rewritten in a canonical form, as a system of first order differential equations:

$$\frac{dy_i}{dt} = g_i(t, y_1, y_2, \ldots, y_n) , \quad i = 1,2,\ldots,n \tag{3}$$

with boundary conditions:

$$\sum_{j=1}^{n} \left\{ h_{1ij}\; y_j(a) + h_{2ij}\; y_j(b) \right\} = h_{oi} , \quad i = 1,2,\ldots,n \tag{4}$$

which may be more conveniently written in matrix notation:

$$\frac{dY}{dt} = G(t,\; Y(t)) \tag{5}$$

$$H_1\, Y(a) + H_2\, Y(b) = H_o \tag{6}$$

where Y, G and H_o are n-vectors and H_1, H_2 are (n x n)-matrices. For the problem defined by (1) and (2), n would of course be 2.

The class of problems which this program is designed to solve includes two important extensions to the basic two point Boundary Value problem described by (3) and (4).

The first is the case of non-linear boundary conditions, which should be expressed in functional form:

$$h_i\; (y_1(a),\; y_2(a), \ldots, y_n(a),\; y_1(b),\; y_2(b), \ldots,\; y_n(b)\;) = 0 \;, i = 1,2,\ldots,n \tag{7}$$

or

$$H(Y(a),\; Y(b)\;) = 0 \tag{8}$$

where H is an n-vector function of Y(a) and Y(b).

The second provides for parametric dependence of the equations, an important special case of this being the Eigenvalue problem. The equations may depend upon a number of parameters μ_i, some of which may be eigenvalues, or unknown parameters to be determined, and some of which may be fixed in the data. A number of eigenvalue parameters may be included, such as the real and imaginary parts of a complex eigenvalue, or the multiple eigenvalue parameters sometimes obtained after separating the variables in a multi-dimensional boundary value problem. For each eigenvalue parameter, a normalizing condition must be included, and for each data determined parameter, an equation must be included to fix that parameter, these conditions being added to the boundary conditions and treated on the same basis. The problem may then be expressed as follows:

$$\frac{dY}{dt} = G(t,\ Y\ (t),\ \mu) \tag{9}$$

with linear boundary conditions:

$$H_1Y(a) + H_2Y(b) + H_3\mu = H_o \tag{10}$$

or non-linear boundary conditions:

$$H(Y(a),\ Y(b),\ \mu) = 0 \tag{11}$$

where μ is a p-vector, H and H_0 are $(n + p)$-vectors, H_1 and H_2 are $(n + p) \times n$ matrices, and H_3 is an $(n + p) \times p$ matrix.

2. DESCRIPTION OF THE METHOD

The method used by the program is a Multiple Shooting method, the theory of which is described by M. R. Osborne (On Shooting Methods for Boundary Value Problems, J.Math.Anal. Appl. 27. 1969) and by H.B.Keller (Numerical Methods for Two-Point Boundary Value Problems, Blaisdell, 1968). The associated initial value problems are solved by a fourth order Runge-Kutta method described by R.England (Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations, Computer Journal, 12, 1969).

The overall interval is broken up into a number of sub-intervals by alternate shooting and matching points, the two end points both being shooting points, and some sub-intervals possibly being null. Estimates are made of the solution at each of the shooting points, and using these as initial values, the differential equation (and in general the variational equations also) are integrated as far as the adjacent matching points. At the matching points, the difference between the two solutions is expressed as a linearized function of the deviations from estimates at the shooting points, and the mismatch in the boundary conditions is similarly expressed. Setting the mismatches to zero, the solution of these linear equations would give the Newton-Raphson correction to the estimates, and for a linear boundary value problem, the correct solution would result. For a non-linear problem, or eigenvalue problem, a number of iterations would be needed, while if the initial estimates were poor, the Newton-Raphson method might not converge. The program makes use of the Harwell library subroutine NS03A to solve the matching equations. This subroutine uses a version of the Marquardt algorithm which largely abandons the Newton-Raphson method in cases where the residuals, or mismatches, do not decrease sufficiently fast, and so enables it to find a solution even from quite bad initial estimates (J.K.Reid, Fortran Subroutines for the Solution of sparse systems of non-linear equations, AERE Harwell Report R7293, H.M.Stationery Office, 1972; R.Fletcher. A modified Marquardt subroutine for non-linear least squares, AERE Harwell Report R6799, H.M.Stationery Office, 1971; D.W.Marquardt, An algorithm for least squares estimation of non-linear parameters, J.SIAM, 11. 1963).

The selection of the sub-intervals, or shooting intervals, can in some cases be difficult, and an analysis of the local stability properties of the equations, performed beforehand, may provide the best distribution of shooting and matching points. However, if the user does not wish to specify the shooting intervals, the program will attempt to choose suitable intervals by itself. It attempts to integrate from each end with steps as large as possible, but stops as soon as a complementary solution (or solution of the variational equations) grows by a factor of C_{fac}*, in the max or L_∞ norm. It then inserts a matching point, and examines the size of the two intervals. Proceeding in the same direction as the larger interval, it obtains an initial estimate at a shooting point coincident with the last matching point, and again integrates until a complementary solution grows by a factor of C_{fac}. The process is repeated until the integrations from the two ends meet.

After the overall interval has been broken up into shooting intervals, the most general problem expressed by equations (9) and (11) may be analysed as follows:

Consider the function $Y(t, u, W, \nu)$ which satisfies

$$\frac{dY}{dt} = G\ (t, Y, \nu) \qquad (12)$$

* Default value is 10.0

where ν is a value of the parameter vector μ, and W is the current initial value at a point $t = u$, expressed by

$$Y(u, u, W, \nu) = W . \tag{13}$$

Defining the Jacobian matrices $M_V(t,u) = \frac{\partial Y}{\partial W}$ and $N(t,u) = \frac{\partial Y}{\partial \nu}$, these matrices satisfy the variational equations:

$$\begin{bmatrix} \frac{dM_V}{dt} & \frac{dN}{dt} \end{bmatrix} = \begin{bmatrix} \frac{\partial G}{\partial Y} & \frac{\partial G}{\partial \mu} \end{bmatrix} \begin{bmatrix} M_V(t,u) & N(t,u) \\ 0_{np} & I_p \end{bmatrix} \tag{14}$$

with the initial conditions:

$$M_V(u,u) = I_n, \quad N(u,u) = 0_{pn} . \tag{15}$$

The system of differential equations (12), (14) may be solved for any given value ν of μ, and from initial conditions (13), (15) at any point $t = u$. Suppose that the shooting points are:

$$u = u_i, \; i = 0, 2, \ldots, q + 1$$

Matching at the internal points:

$$t = t_i, \; i = 1, 3, \ldots, q$$

and imposing the boundary conditions, the algebraic equations to be solved are:

$$Y(t_i, u_{i-1}, W_{i-1}, \nu) = Y(t_i, u_{i+1}, W_{i+1}, \nu), \quad i = 1, 3, \ldots, q \tag{16}$$

$$H(W_o, W_{q+1}, \nu) = 0 . \tag{17}$$

If the solution is:

$$Y_i = W_i + E_i, \; i = 0, 2, \ldots, q+1 \tag{18}$$

$$\mu = \nu + \zeta \tag{19}$$

the Newton-Raphson correction (E_i, ζ) is given by:

$$M_V(t_i, u_{i-1}) E_{i-1} - M_V(t_i, u_{i+1}) E_{i+1} + \left(N(t_i, u_{i-1}) - N(t_i, u_{i+1})\right) \zeta$$

$$= Y(t_i, u_{i+1}, W_{i+1}, \nu) - Y(t_i, u_{i-1}, W_{i-1}, \nu), \quad i = 1, 3, \ldots, q \tag{20}$$

$$\frac{\partial H}{\partial Y(a)} E_o + \frac{\partial H}{\partial Y(b)} E_{q+1} + \frac{\partial H}{\partial \mu} \zeta = - H(W_o, W_{q+1}, \nu) . \tag{21}$$

If $M_v(t_i, u_{i-1}) = M_{o,i-1}, \quad M_v(t_i, u_{i+1}) = M_{oi}$

$N(t_i, u_{i-1}) - N(t_i, u_{i+1}) = N_{oi},$

$Y(t_i, u_{i+1}, W_{i+1}, \nu) - Y(t_i, u_{i-1}, W_{i-1}, \nu) = \Gamma_{oi},$

$i = 1,3,\ldots q$

this may be written in block matrix form

$$\begin{bmatrix} M_{oo} & -M_{o1} & & & & N_{o1} \\ & M_{o2} & -M_{o3} & & & N_{o3} \\ & & & M_{o,q-1} & -M_{oq} & N_{oq} \\ \frac{\partial H}{\partial Y(a)} & & & & \frac{\partial H}{\partial Y(b)} & \frac{\partial H}{\partial \mu} \end{bmatrix} \begin{bmatrix} E_o \\ E_2 \\ -- \\ E_{q+1} \\ \zeta \end{bmatrix} = \begin{bmatrix} \Gamma_{o1} \\ \Gamma_{o3} \\ -- \\ \Gamma_{oq} \\ -H(W_o, W_{q+1}, \nu) \end{bmatrix} . \quad (22)$$

The Jacobian matrix on the left of this equation, together with the residuals on the right, are supplied to the subroutine NSO3A, which determines a series of Newton-Raphson, or appropriately adjusted Marquardt corrections, until the desired accuracy is obtained.

3. THEORETICAL CONSIDERATIONS

The more common numerical integration methods currently used to solve two point boundary value problems may be illustrated by reference to the equation:

$$\frac{d^2y}{dt^2} + \lambda y = 0 . \quad (23)$$

If $\lambda > 0$ the solutions of this equation are oscillatory, and it is usual to use a shooting method in which estimates are made of the initial conditions, and these are then adjusted when the mismatch in the end conditions is known. Such methods are useless when $\lambda < 0$ as any slight error in the initial conditions, or in the integration, will be magnified at the end point, and it will not be possible to perform a stable integration in either direction. A finite difference method must then be used, in which algebraic equations are set up for a large number of function values, and are then solved to give the solution at every mesh point simultaneously. A finite difference method is inadvisable when $\lambda > 0$, as the algebraic matrix is then likely to be singular.

In the Multiple Shooting method, the overall interval is broken up into a number of sub-intervals, which are chosen in such a way as to attempt to limit the growth of errors in any sub-interval. In the two extremes, the method reduces to an ordinary shooting method if only one sub-interval is used, or to a finite difference method if the integration across each sub-interval is performed in one step. Other standard cases, such as shooting from each end and matching in the middle, are also included. The method can be used for all values of λ by choosing suitable sub-intervals, and if λ changes sign in the interval of interest, it is possible to use a number of small sub-intervals where λ may be negative, and large sub-intervals elsewhere. The user need not necessarily specify the sub-intervals, as the program will, if necessary, attempt to choose sub-intervals for itself. However, its method of choosing sub-intervals is not always satisfactory. In equation (23), if λ is a function of an eigenvalue, an estimate of that eigenvalue must be made by the user. If the estimate results in too large a value of λ, the potential growth of the error will not be detected by the program, which will choose its sub-intervals too large. Three methods may be available for avoiding this difficulty. If the equations have a particularly simple form such as (23), it may be possible to make estimates on the safe side, such as estimating λ rather smaller than the value being sought. It may be necessary to perform more detailed analysis, from which suitable sub-intervals can probably be determined. Finally, a first run of the program will almost certainly improve even poor estimates, and the program may be run again from improved estimates, when the choosing of intervals will give more satisfactory results.

4. PROGRAMMING LANGUAGE

The program is written in FORTRAN IV and is available on the IBM 360 - 370* series and the ICL 4/70 computers.

The routines of DD03 are the same for both systems. However the Harwell Library Subroutine NS03A, and the others which it calls, use some extensions to FORTRAN IV peculiar to the 360 implementation, in particular that allowing fairly general integer expressions to be used as array subscripts. These subroutines have been converted for use on the 4/70.

The subroutine DD03 is available only in its double length version on both systems since a number of test examples which have been solved successfully using double length arithmetic failed due to rounding error problems when single length was employed.

The various library subroutines contain some input-output statements for printing intermediate results and error diagnostics during the iteration, but otherwise all input-output statements are concentrated in the highest level subroutine SETDEF, and in the small main program which must be provided by the user to define the problem size.

Certain facilities of IBM 360 and ICL 4/70 Fortran, which have been used, should perhaps be noted. 2 byte integers are used to save space in some places. Namelist input is used for all the data, which makes it easy to set default values, and allows the data to be in a fairly free format. Data statements are used to initialize those variables

* The Harwell Subroutine Library Specification is reproduced in Section 11.

which are not arrays. Multiple entry subroutines are used, where the first entry is usually used to initialize the addresses and values of some variables, as well as the dimensions of some of the dynamic arrays - though this only matters for those with more than one dimension. In a number of places, where the symbolic arguments of a subroutine are arrays, the actual arguments used are array elements, which causes the subroutine to use for its array, that part of the whole array which begins at the element specified. In particular, the work space array, TU, provided by the user, is broken up into smaller arrays in this way, and some parts of the array are used as 2 byte integer arrays.

5. INPUT TO THE PROGRAM

The most important input required by the program is a set of one or two subroutines which describe the problem as defined by one of the sets of equations (5), (6), or (5), (8) or (9), (10) or (9), (11), together with a small main program which declares arrays of sufficient size for the problem, and reads in initial values to some of the arrays.

The suggested form of the main program is as follows, where the small letters must be replaced by actual values for the problem in hand:

```
REAL*8 U(p),R(r),YR(n,r),S(s),DH(n+p,2n+p+1),TU(w),ZET,FACT,CFAC
COMMON/ DDO3CD/ZET,FACT,MAXF,LP,CFAC
NAMELIST/ARRAYS/DH,S,R,YR,U,TU,CFAC
CALL SETDEF(n, n+p, 2n+p+1,w,r,DH,TU,R,YR)
READ(5,ARRAYS)
CALL PROGC(S,U)
STOP
END
```

Here n and p are parameters of the problem; s is the maximum number of points at which the solution may be required; r is a number greater than s, than the maximum number of shooting points, and than the number of points at which an estimate of the solution may be given; and w is the number of double words of work space available in the array TU. If q shooting points are used (the q of section 2 was almost twice the value of this q), then w need not be more than:

$$q(2 + 9\tfrac{1}{2}n + 10n^2 + 4\tfrac{1}{2}np) + 1 + 7n + 5n^2 + 9\tfrac{1}{2}p + 3\tfrac{1}{2}p^2 + 14np$$

while about $\frac{2}{3}$ of this will often be sufficient.

A subroutine G is also essential to define the set of differential equations given by (5) or (9). Where it is reasonably simple, the program works better if subroutine G also provides values of the analytic derivatives, or Jacobian matrices $\frac{\partial G}{\partial Y}$ (and $\frac{\partial G}{\partial \mu}$). A data item, YMUST, described below, must indicate whether this has been done. The equations (5), (6) or (5), (8) or (9), (10) or (9), (11) must first be scaled so that all the variables in the vectors Y and μ are of the same order of magnitude. Subroutine G should then take the following form:

SUBROUTINE G(T,Y,U,FG,DG,N)

REAL*8 T,Y(N),U(p),FG(N),DG(N,n + p)

Statements to determine the vector FG $= \frac{\partial Y}{\partial t} = G(T, Y)$

or FG $= \frac{\partial Y}{\partial t} = G(T, Y, U)$

Optionally statements to determine the elements of the Jacobian matrix DG $= \left[\frac{\partial G}{\partial Y}\ \frac{\partial G}{\partial \mu}\right]$

RETURN

END

If the problem has linear boundary conditions given by (6) or (10), a default subroutine H[*] is provided with the program for dealing with them, and the boundary conditions are defined in the data. If the problem has non-linear boundary conditions given by (8) or (11), then a subroutine H must be written to define the vector function H. It should have the following form:

SUBROUTINE H(YA, YB, U, FH, DH, NP)

REAL*8 YA(n),YB(n),U(p),FH(NP),DH(NP,2n + p + 1)

Statements to determine the vector FH = H(YA,YB)

or FH = H(YA,YB,U)

Statements to determine those elements of DH $= \left[\frac{\partial H}{\partial Y(a)}\ \frac{\partial H}{\partial Y(b)}\ \frac{\partial H}{\partial \mu}\right]$ which are functions of Y(a), Y(b) and μ. Any elements of DH which are constant may be initialized in the data and need not be set in subroutine H. The last column of DH is not used except by subroutine H, and may be used to communicate parameter values from the data to subroutine H.

RETURN

END

The main program and one or both subroutines must be compiled and the rest of the program composed with them. If the main program takes the form suggested, then the data must be as follows, where the usual rules for NAMELIST input apply, i.e. data is punched in columns 2 to 80, the first card having an & and the name of the namelist, the last card having &END, and intervening cards having an assignment to a program variable or array, terminated by a comma, and in the case of an array, having individual elements separated by commas. Assignments of numerical values may be made to any of the variables in the

[*] listed in Appendix A

namelists, but others may be left with their default values.

&VALUES

any of the following:

RI = number of values of t at which initial estimates of Y(t) are given ($\leqslant r$; default 0; is this is zero, then zero will be taken as an initial estimate of μ and of Y(t) everywhere; if it is negative, zero will be taken as an initial estimate of μ, but IABS(RI) estimates of Y(t) will be used; if it is positive, the estimate of μ and RI estimates of Y(t) will be used; linear interpolation between the given values will be used to obtain estimates at the shooting points),

SI = number of values of t at which the solution Y(t) is required ($\leqslant s$; default 0; if this is zero, the solution is given at the shooting points actually used by the program),

A = a (default 0.0),

B = b (default 1.0; $b > a$)

ZETA = root-mean-square error bound required of the iteration process used to solve the potentially non-linear algebraic equations (default 10^{-5}),

EPSI = error bound for each element of Y(t), used by the Runge-Kutta integration routine as a target for the integration error across each shooting interval (default -10^{-4}; a negative value indicates a relative error bound; if this is set to zero it is replaced by the value of ZETA; to obtain differential accuracies for the different elements of Y(t) the equations have to be rescaled),

YMUST = increment in Y and μ for use in obtaining numerically the derivatives in the Jacobian matrices $\frac{\partial G}{\partial Y}$ and $\frac{\partial G}{\partial \mu}$ (default 0.0; subroutine G must provide analytic values of the derivatives if and only if this is zero),

MAXFUN = maximum number of integrations allowed (default 10),

LPRINT = number of iterations between printouts (default 0; a value of zero gives no intermediate output, while a negative value gives more complete details than a positive value),

LP = stream number for the printed output (default 6),

DI = number of shooting points specified ($\leqslant r$; default 0; if this is zero, or if the work space in array TU is insufficient, or if the specified points are not in ascending order, then the program chooses shooting intervals for itself),

&END

&ARRAYS

any of the following:

DH(i, j) = - - - (default 0.0; assignments of this form may be used to set any elements of DH which are constant, and so need not be determined by subroutine H, or parameter values in the last column of DH; in the case of linear boundary conditions (6) or (10), DH is the composite matrix consisting of $[H_1\ H_2\ H_3\ H_0]$, its last column H_0 being the right hand side of the boundary condition equations, and all non-zero elements are constant and must be set),

S = list of SI specified tabulation points, or values of t at which the solution Y(t) is required,

R = list of IABS(RI) values of t, arranged in order from a to b, at which estimated values of Y(t) are given,

YR = list of estimated values of Y(t) at the values of t specified in R, where all the elements of one vector Y(t) are given before those of the next,

U = estimated value of μ, used only if RI is positive,

TU = list of Q shooting points TU(1), TU(3), ..., TU(2Q - 1) interlaced with Q - 1 matching points TU(2), TU(4), ..., TU(2Q - 2),

CFAC = factor allowed for growth of complementary solutions (default = 10.0),

&END

All this data is read from stream 5.

6. OUTPUT FROM THE PROGRAM

The call to subroutine SETDEF performs only the initialization of some addresses and default values, and the reading of the namelist VALUES. PROGC is a side entry to the same routine, and immediately it is called, details of the input data are written to the output stream LP :

```
NUMBER OF DEPENDENT VARIABLES Y: N = n

NUMBER OF BOUNDARY CONDITIONS: N + P = NO = n + p

NUMBER OF VARIABLES INVOLVED IN BOUNDARY CONDITIONS: 2N + P + 1 = NN = 2n + p + 1

NUMBER OF WORDS OF WORK SPACE PROVIDED: ITU = w

DIMENSION OF EIGENVALUE MU: P = p

MAXIMUM STORAGE FOR TABULATION POINTS OR ESTIMATION POINTS: R = r

NUMBER OF ESTIMATED VALUES OF Y (POSITIVE IF MU IS ESTIMATED): RI = RI

NUMBER OF SPECIFIED TABULATION POINTS: SI = SI

VALUES OF INDEPENDENT VARIABLE T AT BOUNDARY POINTS: A = a
                                                     B = b

DEFAULT VALUES FOR JACOBIAN MATRIX DH

OR COEFFICIENTS OF BOUNDARY CONDITIONS H1, H2, H3, HO
```

This is followed by the matrix DH, the list of

SPECIFIED TABULATION POINTS (if SI is positive), the

ESTIMATED VALUES OF Y (if RI is non-zero), the

ESTIMATED VALUE OF EIGENVALUE MU (if p and RI are positive),

NUMBER OF ITERATIONS BETWEEN PRINTOUTS: LPRINT=

MAXIMUM NUMBER OF INTEGRATIONS ALLOWED: MAXFUN=

INCREMENT IN Y AND MU FOR DIFFERENTIATION OF THE FUNCTION G: YMUST=

ERROR BOUND ON SOLUTION: ZETA=

ERROR BOUND FOR THE DEPENDENT VARIABLES Y DURING INTEGRATION: EPSI=

DI SPECIFIED SHOOTING POINTS (and matching points from TH).

After the list of shooting and matching points has been written, the subroutine DD03AD is called, to carry out the solution of the problem defined. The specification of this subroutine appears in section 11 of this report. The main output produced during the solution is that written by subroutine NS03AD after every LPRINT iterations, if LPRINT is non-zero. Output is written at the first and last iteration and at every IABS(LPRINT)th iteration in between, as described in the Harwell library subroutine specification of NS03A (parameter IPRINT). If LPRINT is negative, the following vectors are output:

the current solution X, which consists of the values of Y at the shooting points and the values of μ.

the current residual vector, which is the vector on the right hand side of equation (22), consisting of the mismatches in Y at the matching points and the mismatches in the boundary condition equations (6), (8), (10) or (11).

the current vector V, which gives an approximation to the derivates of $\frac{1}{2}$S with respect to the elements of X, where S is the sum of squares of the residuals or mismatches.

This output is in addition to the following summary which is printed when LPRINT is positive:

the number of iterations and integrations (sometimes more than one integration may be performed in one iteration).

the Marquardt parameter λ.

the sum of squares of residuals S.

the Euclidean norms (roots of sums of squares) of the vector X, the last change made to it, and the vector V.

Output is also written by certain library subroutines to report some error conditions which may arise. In particular, if the work space w is not large enough, if more than

MAXFUN integrations are required, if b is not greater than a, or if the iteration process fails to converge to a solution, then a message is written and no further improvement to the solution is carried out. Error messages may be written by MA17 if overwriting of the data has occurred, or if the overall Jacobian matrix of equation (22) is singular, which may occur if the problem is not properly posed, if the initial estimates are unsuitable, or if the increment and error bound YMUST and EPSI are unsuitable. DDO3AD also writes a message if there is insufficient space for the specified number of shooting points DI, or if the shooting and matching points are not in ascending order, and it reverts to choosing intervals for itself.

Whether or not an error has occurred, the best result obtained (which may be only the initial estimates, if the work space w is insufficient) is then tabulated under two headings:

TABULATED VALUE OF Y

TABULATED VALUE OF EIGENVALUE MU

the latter being printed only if p is positive.

Following this, in order to give information on the working of the program, the following information is printed:

DI ACTUAL SHOOTING POINTS (if there was enough work space to perform a complete integration)

w WORDS OF WORKSPACE USED

and control is then returned to the main program at the statement STOP.

7. SAMPLE PROBLEM

The problem given here arises from the hydrodynamic flow between two infinite rotating discs. It is given in this form by M.R. Osborne (On Shooting Methods for Boundary Value Problems, J.Math.Anal.Appl., 27, 1969) but the variables have already been appropriately transformed and rescaled from the original problem (G.N. Lance and M.H. Rogers, The symmetric flow of a viscous fluid between two infinite rotating discs, Proc.Roy.Soc. 266, (1962), pp. 109 - 121):

$$\frac{dx_1}{dt} = -2x_2$$

$$\frac{dx_2}{dt} = x_3$$

$$\frac{dx_3}{dt} = x_1x_3 + x_2^2 - x_4^2 + k$$

$$\frac{dx_4}{dt} = x_5$$

$$\frac{dx_5}{dt} = 2\ x_2x_4 + x_1x_5$$

$$x_1(0) = x_2(0) = 0\ , \qquad x_4(0) = 1$$

$$x_1(b) = x_2(b) = 0\ , \qquad x_4(b) = s\ . \tag{24}$$

b and s are constants depending on the separation of the discs and their speeds of rotation, while k is an unknown parameter to be determined along with the functions x_1, x_2, ..., x_5. (For physical interpretations see Lance and Rogers, as quoted above).

The routines required to solve this problem are as follows:

```
      REAL*8 K,R(10),YR(5,10),S(10),DH(6,12),TU(3464),ZET,FACT,CFAC
      COMMON /DD03CD/ZET,FACT,MAXF,LP,CFAC
      NAMELIST/ARRAYS/DH,S,R,YR,K,TU,CFAC
      CALL SETDEF(5,6,12,3464,10,DH,TU,R,YR)
      READ(5,ARRAYS)
      CALL PROGC(S,K)
      STOP
      END

      SUBROUTINE G(T,X,K,FG,DG,N)
      REAL*8 T,X(N),K,FG(N),DG(N,6)
      FG(1) = - 2.0*X(2)
      FG(2) = X(3)
      FG(3) = X(1)*X(3) + X(2)*X(2) - X(4)*X(4) + K
      FG(4) = X(5)
      FG(5) = 2.0*X(2)*X(4) + X(1)*X(5)
      DO 1  J = 1,6
      DO 1  I = 1,N
1     DG(I,J) = 0.0
      DG(1,2) = - 2.0
      DG(2,3) = 1.0
      DG(3,1) = X(3)
      DG(3,2) = 2.0*X(2)
      DG(3,3) = X(1)
      DG(3,4) = - 2.0*X(4)
      DG(3,6) = 1.0
      DG(4,5) = 1.0
      DG(5,1) = X(5)
DG(5,2) = 2.0*X(4)

DG(5,4) = 2.0*X(2)

DG(5,5) = X(1)

RETURN

END
```

To solve a case where b = 18 and s = 0.5, a suitable set of data is as follows:

```
&VALUES

RI = - 2,

B = 18.0,

ZETA = 1E-6,

MAXFUN = 40,

LPRINT = 1,

DI = 10,

&END

&ARRAYS

DH(1,1) = 1.0,

DH(2,2) = 1.0,

DH(3,4) = 1.0, DH(3,12) = 1.0,

DH(4,6) = 1.0,

DH(5,7) = 1.0,

DH(6,9) = 1.0, DH(6,12) = 0.5,

R = 0.0, 18.0,

YR = 0.0, 0.0, 0.0, 1.0, 0.0,

     0.0, 0.0, 0.0, 0.0, 0.0,

TU = 0.0, 2.0,

     2.0, 4.0,

     4.0, 6.0,

     6.0, 8.0,

     8.0, 10.0,

     10.0, 12.0,

     12.0, 14.0,

     14.0, 16.0,

     16.0, 18.0,

     18.0,

&END
```

With this data, the program uses the same shooting intervals as were used by M.R. Osborne, as quoted above, and should converge to the same solution with only 11 integrations, as compared with the 26 quoted by Osborne, using a less sophisticated method for solving the algebraic equations. The comparison gives some idea of the operation of the program, but is not entirely valid, as the initial estimates and final accuracy are defined differently. The output for this case is reproduced in Appendix C.

8. CONVERGENCE AND ESTIMATION

Some of the data items required by the program are not immediately obvious from the mathematical definition. In this respect, the items R, YR, U, EPSI, TU are discussed here. R, YR, U constitute the initial estimate of the solution. It may be natural at a first attempt to provide no estimate, or some properties of the solution may be conveniently known. In the sample problem, a first attempt was made with no estimate, which meant that zero values were used, as by M.R. Osborne. However, the estimate with all the functions equal to zero was an exact solution of the differential equations, and the integration routine integrated over each sub-interval in a single step without error. In this case, because of the zero solution, many of the partial derivatives in the Jacobian matrix, which are in general non-zero, give zero values, and in some cases this has given rise to a singular matrix. Therefore it is important that the initial estimates are not special values, which may give rise to a particular simple solution. Even for a linear problem, which does not in general require initial estimates, the use of zero estimates may give rise to exaggeratedly large integration steps, and thereby to excessively large shooting intervals, and the required accuracy may then not be obtainable.

It should be noted that, on return from the call to PROGC, the last estimate of the solution (at the tabulation points) is contained in R, YR, U. If it is required to find another solution with different parameter values, or to find a more accurate solution by decreasing the error bounds, it is sufficient to call PROGC again, when the last solution will be used as an estimate. In that case, the same shooting intervals will be used again. If it is required that new shooting intervals be chosen, then SETDEF must first be called to read the Namelist VALUES from the data, and DI must be reset to zero in that data.

The error bound ZETA is used in such a way that a normal return is made if, on any iteration, both the root-mean-square of the residuals or mismatches (on the right-hand side of equation (22)), and the root-mean-square of the changes made to Y and μ, are less than ZETA. Therefore, in order that EPSI should provide the same order of error bound, its value, in absolute terms, should be approximately equal to ZETA. (If a negative value of EPSI is specified, then the actual error bound used for y_i, in integrating across any shooting interval, is $|EPSI|(|EPSI| + |y_i|)$ where a local value of y_i is used.) However, a first attempt, using the desired values of ZETA and EPSI, may encounter an integration which is excessively difficult, and uses up too much time without completing an iteration. In such cases, it may be well first to improve the initial estimates by using larger error bounds, and then to decrease them and call PROGC again. However, if excessively large values of EPSI are used, the integration routine may take exaggeratedly large integration steps once more, and produce a small error estimate although the actual error is large.

If shooting intervals are not specified, the program attempts to choose them for itself, but its process for choosing them is not always satisfactory. In particular, trouble may arise if the properties of the equation (the nature of the complementary solutions) vary considerably over the interval (a, b) or between the initial estimate and the desired solution. If the variation is across the interval, it may be necessary to analyse the complementary solutions, and so decide what size of shooting interval is possible in each direction in each part of the overall interval, and so set up appropriate shooting intervals in array TU. If the variation is due to a poor initial estimate, it should usually be possible to obtain good results by first improving the initial estimate using large values of ZETA and EPSI, and then resetting DI = 0, reducing ZETA and EPSI and calling PROGC again.

9. EXTENSIONS TO THE PROBLEM CLASS

Certain problems which are not immediately seen to be of the form solved by this program, may nevertheless be transformed in such a way that it will solve them. The particular difficulties discussed here are boundary conditions at infinity, and other singularities in the natural specification, complex variables in the mathematical description, and the treatment of parameters or constants upon which the solution depends. In the case of singularities, so many types may occur that a general discussion is impossible, but the difficulties mentioned above will be illustrated by two examples.

The first is a very simple one dimensional Schrödinger equation, whose dominant eigenfunction is known, and which illustrates the treatment of a boundary condition at infinity. The problem was suggested by Dr R.C. Grimm, recently of Culham Laboratory.

$$- \frac{d^2}{dx^2} \psi(x) + 20 \tanh^2(x)\psi(x) = E\psi(x)$$

$$\psi(o) = \psi(\infty) = 0 \;. \tag{25}$$

The known solution to this problem is E = 11, $\psi(x) = \alpha \operatorname{sech}^3(x)\tanh(x)$ for any constant α. Writing $y = \frac{d\psi}{dx}$ we have

$$\frac{dy}{dx} = (20 \tanh^2(x) - E)\psi$$

$$\frac{d\psi}{dx} = y \tag{26}$$

which is in the form (9).

As $x \to \infty$, $\tanh^2(x) \to 1$ and so the asymptotic solution of (26) has the form

$$y = A \exp(x\sqrt{20 - E}) + B \exp(-x\sqrt{20 - E})$$

$$\psi = \frac{A}{\sqrt{20 - E}} \exp(x\sqrt{20 - E}) - \frac{B}{\sqrt{20 - E}} \exp(-x\sqrt{20 - E}) \;. \tag{27}$$

For most boundary conditions at infinity, it should be possible to find such an asymptotic solution as $t \to \infty$, and to apply the boundary condition to it, giving in this case A = 0 (20 > E). From this we conclude the following boundary condition as $x \to \infty$:

$$y(x) + \sqrt{20 - E}\, \psi(x) \sim 0 \tag{28}$$

while we could not use $\psi(x) \sim 0$ as this would never be satisfied exactly by a non-zero solution. It remains to decide by what finite value of x we can approximate infinity, and in this case, the asymptotic solution is already very closely satisfied for x = 10, and the new asymptotic boundary condition can be applied at that point. An additional boundary, or normalizing, condition is also required to define the problem completely, and in order to ensure a non-zero solution (indeed a solution with $\alpha = 1$) a suitable condition is y(o) = 1 . The complete set of boundary conditions of the form (11) is then:

$$y(o) = 1$$

$$\psi(o) = 0$$

$$y(10) + \sqrt{20 - E}\, \psi(10) = 0 \;. \tag{29}$$

Again for this problem, attention should be given to the initial estimates, as if no estimates are given, the program will first find the zero solution and integrate across the whole interval in one step, while in order to find a good solution, some 10 to 20 shooting intervals are needed. In fact, zero estimates should not be given at either end, as if the estimates at x = 10 are zero, the program will again integrate all the way back in one step, and use only two shooting intervals. Suitable estimates are:

$$\left.\begin{array}{ll} y(o) = 1\,, & \psi(o) = 0 \\ y(1) = 0\,, & \psi(1) = 1 \\ y(10) = -3\times 10^{-12}\,, & \psi(10) = 10^{-12} \\ E = 10 & \end{array}\right\} \qquad (30)$$

and with these estimates, four figure accuracy should be obtainable in around 10 iterations. This problem was solved using an earlier version of the program with different characteristics. So no exact details are available as to the performance of this program on this problem.

The second example was suggested by Dr C.N. Lashmore-Davies, of Culham Laboratory (Stabilization of a low-density plasma in a simple magnetic mirror of feedback control, J. Phys. A: Gen. Phys. 4, 1971).

It illustrates a singularity of the type occurring in Bessel's equation (for some particular parameter values it may be reduced to Bessel's equation), as well as methods for treating complex variables and parametric dependence of the problem. The mathematical expression of the problem is as follows:

$$\left(1+\frac{\omega_{pi}^2}{\Omega_i^2}N(x)\right)\frac{d^2\varphi}{dx^2}+\left[\left(1+\frac{\omega_{pi}^2}{\Omega_i^2}N(x)\right)\frac{1}{x}+\frac{\omega_{pi}^2}{\Omega_i^2}\frac{dN}{dx}\right]\frac{d\varphi}{dx}-\frac{m^2}{x^2}\left(1+\frac{\omega_{pi}^2}{\Omega_i^2}N(x)\right)\varphi$$
$$+\,m^2\frac{\omega_{pi}^2}{\Omega_i}\frac{\omega^*}{\omega(\omega+m\omega^*)}\frac{1}{x}\frac{dN}{dx}\varphi = 0\,, \qquad (31)$$

$$\frac{d}{dx}\varphi(1+)-\frac{d}{dx}\varphi(1-)-\frac{\omega_{pi}^2}{\Omega_i^2}N(1-)\frac{d}{dx}\varphi(1-) = m^2\frac{\omega_{pi}^2}{\Omega_i}\frac{\omega^*}{\omega(\omega+m\omega^*)}N(1-)\,\varphi(1) \qquad (32)$$

$$\frac{d}{dx}\varphi\left(\frac{b}{a}+\right)-\frac{d}{dx}\varphi\left(\frac{b}{a}-\right) = \delta\,\varphi(1) \qquad (33)$$

$$\varphi(x)\to 0 \quad \text{as} \quad x\to\infty \qquad (34)$$

$$-\infty < \varphi(o) < +\infty \qquad (35)$$

where ω_{pi}, Ω_i, ω, ω^* are parameters whose approximate ratios are $\omega_{pi}^2 = O(\Omega_i\,\omega^*) \ll \Omega_i^2$, $\Re(\omega) = O(-\tfrac{1}{2}\omega^*)$, m is an integer parameter with small value, a , b , δ are parameters such that $0 < a < b$, and $N(x)$ is a known function which is zero for $|x| > 1$, and for which the particular case $N(x) = 1 - x^2$, $(|x| < 1)$ is of interest. It is desired to find the relationship between the complex eigenvalue ω and the parameter δ, for given values of the ratios $\frac{b}{a}$, $\frac{\omega_{pi}}{\Omega_i}$, $\frac{\omega_{pi}^2}{\Omega_i\,\omega^*}$, and of m .

Setting $$p^2 = -\,2m^2\,\frac{\omega^2_{pi}}{\Omega_i\ \omega^*}\ \frac{1}{\frac{\omega}{\omega^*}\left(\frac{\omega}{\omega^*}+m\right)}$$

$$q^2 = \left(\frac{\omega_{pi}}{\Omega_i}\right)^2$$

$$r^2 = \frac{\omega_{pi}^{\ 2}}{\Omega_i\ \omega^*} \tag{36}$$

the problem reduces to :

$$\frac{d^2\varphi}{dx^2} + \left[\frac{1}{x} + \frac{q^2}{1+q^2N(x)}\ \frac{dN}{dx}\right]\frac{d\varphi}{dx} - \left[\frac{m^2}{x^2} + \frac{p^2}{2x(1+q^2N(x))}\ \frac{dN}{dx}\right]\ \varphi = 0\ . \tag{37}$$

$$\frac{d}{dx}\ \varphi(1+) - (1 + q^2N(1-))\ \frac{d}{dx}\ \varphi(1-) + \tfrac{1}{2}p^2N(1-)\varphi(1) = 0\ . \tag{38}$$

$$\frac{d}{dx}\ \varphi\left(\frac{b}{a}+\right) - \frac{d}{dx}\ \varphi\left(\frac{b}{a}-\right) - \delta\varphi(1) = 0\ . \tag{39}$$

$$\varphi(x) \to 0 \quad \text{as} \quad x \to \infty \tag{40}$$

$$-\infty < \varphi(0) < +\infty\ . \tag{41}$$

Since $N(x) = 0$ for $|x| > 1$, the solution for $x > 1$ may be performed analytically, and gives:

$$\varphi(x) = \frac{1}{2}\left[\left(\varphi(1) + \frac{1}{|m|}\ \frac{d}{dx}\ \varphi(1+)\right)\ x^{|m|} + \left(\varphi(1) - \frac{1}{|m|}\ \frac{d}{dx}\ \varphi(1+)\right)\ x^{-|m|}\right] \text{ for } 1 \leqslant x \leqslant \frac{b}{a}$$

$$\tag{42}$$

$$= \frac{1}{2}\left[\left(\varphi(1) + \frac{1}{|m|}\ \frac{d}{dx}\ \varphi(1+)\right)\left(\frac{b}{a}\right)^{2|m|} + \left(\varphi(1) - \frac{1}{|m|}\ \frac{d}{dx}\ \varphi(1+)\right)\right]\ x^{-|m|} \quad \text{for} \quad x \geqslant \frac{b}{a}\ .$$

Applying conditions (38) and (39) this gives the condition at $x = 1$:

$$\left[\delta\ \left(\frac{a}{b}\right)^{|m|-1} + |m| - \tfrac{1}{2}p^2N(1-)\right]\ \varphi(1) + \left(1 + q^2N(1-)\right)\ \frac{d}{dx}\ \varphi(1-) = 0\ . \tag{43}$$

In general, $N(x) = 1 + 0(x^2)$ as $x \to 0$, and performing an asymptotic solution as $x \to 0$,

$$\varphi \sim Ax^{|m|} + Bx^{-|m|} \quad \text{if} \quad m \neq 0\ . \tag{44}$$

Applying the condition (41) to this gives $B = 0$, or the boundary condition

$$\varphi(0) = 0\ . \tag{45}$$

Finally, defining $y(x) = (1 + q^2N(x))\ \frac{d\varphi}{dx}$ for $|x| < 1$, all the singularities can be eliminated to give:

$$\begin{aligned}\frac{dy}{dx} &= \left[\frac{m^2}{x^2}\left(1 + q^2N(x)\right) + \frac{p^2}{2x}\frac{dN}{dx}\right]\varphi - \frac{1}{x}\,y \quad \text{for} \quad x \neq 0\\ &= 0 \quad \text{for} \quad x = 0 \quad \text{if} \quad |m| \neq 2\\ &= 2A \quad \text{for} \quad x = 0 \quad \text{if} \quad |m| = 2 \qquad (46)\\ \frac{d\varphi}{dx} &= \frac{1}{1 + q^2N(x)}\,y\end{aligned}$$

with boundary conditions :

$$\varphi(0) = 0$$

$$\left[\delta\left(\frac{a}{b}\right)^{|m|-1} + |m| - \tfrac{1}{2}p^2N(1-)\right]\varphi(1) + y(1) = 0 \qquad (47)$$

and a normalizing condition chosen to be compatible with the exact Bessel function solution $\varphi(x) = J_m(px)$ which arises when $N(x) = 1 - x^2$ and $q^2 = 0$ (which also gives rise to $A = \frac{1}{8}p^2$ in (46)) :

$$\varphi(1) = J_m(p) \,. \qquad (48)$$

The next problem is that the eigenvalue, ω, is in general a complex number, and must be split into its real and complex parts as follows, and related to the variables appearing in (46), (47) and (48) :

$$\omega = \omega^*(u + iv) \qquad (49)$$

$$p^2(u^2 - v^2 + um) = -2m^2r^2 \,, \qquad v(2u + m) = 0 \qquad (50)$$

the last arising from the definition of p^2 in (36).

The following variables now appear in the problem, either as eigenvalues to be determined, or as parameters to be varied to give different solutions:

$$\delta\left(\frac{a}{b}\right)^{|m|-1}, \; m, \; p, \; q^2, \; r^2, \; u, \; v$$

and it is convenient to make the vector μ, appearing in (9) and (11) consist of these seven variables. The required nine boundary conditions of the form (11) are then made up of (47), (48), (50) and four conditions:

$$\begin{aligned}&\alpha\left(\delta\left(\frac{a}{b}\right)^{|m|-1} - d_o\right) + (1-\alpha)(p - p_o) = 0\\ &m - m_o = 0\\ &q^2 - q_s = 0 \qquad\qquad (51)\\ &r^2 - r_s = 0\end{aligned}$$

where α (= 0 or 1), d_o, p_o, m_o, q_s, r_s may be provided as data through the last column of DH. If m, q^2, r^2 are specified through m_o, q_s, r_s, then by varying α, the program can be used to specify $\delta\left(\frac{a}{b}\right)^{|m|-1} = d_o$ and find p, u, v, or to specify $p = p_o$ and find $\delta\left(\frac{a}{b}\right)^{|m|-1}$, u, v. A number of cases were successfully solved using the earlier version of the program, with $N(x) = 1 - x^2$, $m = 1$, $q^2 = 0$, $r^2 = 1$ (values of p_o in the range 1 to 6, and two values of d_o were used). Attention should again be given to the initial estimates, which should not be zero, for the same reason as before. Also, so that none of the partial derivatives of the boundary conditions with respect to the parameters should accidentally be zero, none of the following expressions should be zero at the initial estimate:

$$\varphi(1),\ p,\ u^2 - v^2 + um,\ 2u + m,\ v\ .$$

Suitable estimates might be those compatible with $\varphi(x) = J_m(px)$ for some typical value of p (such as p = 3) which is not a root of J_m, with $\delta\left(\frac{a}{b}\right)^{|m -1} = 0$, $u = 0$, $v = 1$. When specifying $\delta\left(\frac{a}{b}\right)^{|m|-1} = d_o$, more attention should be given to the estimate of p, as multiple solutions are possible.

10. INDEX TO VARIABLES AND SUBROUTINES

The first subroutine described here is the default subroutine H, which is only used for problems with linear boundary conditions given by (6) or (10), and determines the difference between the left and right hand sides of those equations. The variables used are as follows:

SUBROUTINE H:

C - an integer equal to $n+p$

N - the integer n

Q - an integer used for counting the equations

S - an integer used for counting the terms in the equations

DH - the composite matrix $[H_1\ H_2\ H_3\ H_0]$

FH - the vector of residuals (result of the subroutine)

MU - the vector of eigenvalue parameters μ

NP - an integer equal to $n+p$

PI - the integer p

QH - an integer used for accessing H_0, the right hand side of equations (6) or (10).

RH - an integer used for accessing H_1, the coefficient of Y(a) in equations (6) or (10).

SH - an integer used for accessing H_2 and H_3, the coefficients of Y(b) and μ in equations (6) or (10)

YA - the vector Y(a)

YB - the vector Y(b)

The next subroutine described is the highest level subroutine SETDEF with entry PROGC, which performs the principal functions of input and output for the program.

SUBROUTINE SETDEF:

A - the boundary point a

B - the boundary point b

C - an integer equal to $n+p$ for the use of the default subroutine H

G - the subroutine defining the differential equations (5) or (9)

H - the subroutine defining the boundary conditions (6), (8), (10) or (11)

I - an integer for counting the elements of Y(t) in either the estimated or the tabulated solution

J - an integer index used for various purposes

K - an integer index used for various purposes

L - the output stream number LP

M - the input stream number 5

N - the integer n for use by the default subroutine H

R - the list of values of t at which the solution Y(t) is either estimated or tabulated

S - the list of values of t at which the solution Y(t) is required

Y - the subroutine entry for performing interpolation between the estimated values of Y(t)

DH - the Jacobian matrix of the function H with respect to its arguments, or the composite matrix $[H_1\ H_2\ \ H_3\ H_0]$

DI - an integer equal to the number of shooting points

LP - the output stream number

MU - the vector of eigenvalue parameters μ

NI - the integer n

NN - an integer equal to $2n+p+1$ or the number of columns in DH

NO - an integer equal to $n+p$ or the number of rows in DH

PI - the integer p

QI - an integer giving the maximum number of tabulation or estimation points

RI - an integer giving the number of values of t at which the solution Y(t) is either estimated or tabulated

SI - an integer giving the number of values of t at which the solution Y(t) is required

TU - a work space array, of which the first 2 × DI - 1 elements give the shooting and matching points

YR - the array of the estimated or tabulated values of the solution Y(t)

ITU - the dimension of the work space array TU

ZET - a default value for ZETA if this is reset to zero

EPSI - error bound for the elements of Y(t) during integration

FACT - the ratio of EPSI to ZETA

MAXF - a default value for MAXFUN

SETY - the subroutine for initializing the interpolation routine Y

ZETA - error bound on the residuals of the solution

DD03AD - the principal entry to the subroutine which solves the problem

DD03BD - a side entry for extracting values of the solution Y(t)

YMUST - increment in Y and μ for differentiation of the function G

LPRINT - the number of iterations between printouts

MAXFUN - the maximum number of integrations

VALUES - the namelist for reading all scalar input

The subroutine SETY with entry Y is used to interpolate between the estimated values of Y(t), or to set the estimates to zero if none are given.

SUBROUTINE SETY:

A - the boundary point a

B - the boundary point b

R - the list of values of t at which the solution Y(t) is estimated

T - the value of t at which an estimate of Y(t) is required

U, V - work variables for the interpolation

FY - the result of the interpolation, or estimate of Y(T)

II - an integer used for counting the values of t in the list R

JJ - an integer used for counting the elements of Y(t) in the estimate FY

NI - the integer n

RI - an integer giving the number of values of t at which the solution Y(t) is estimated

YR - the array of estimated values of the solution Y(t)

The subroutine DD03AD with entry DD03BD is the highest level subroutine of a package which actually solves the problem. Its principal function is the allocation of the work space array TU into smaller arrays for use by the other routines.

SUBROUTINE DD03AD:

A - the boundary point a

B - the boundary point b

G - the subroutine defining the differential equations (5) or (9)

H - the subroutine defining the boundary conditions (6), (8), (10) or (11)

I - an integer used for counting elements of the array TU

J - an integer giving the size of various sub-arrays of TU

K - an integer used for extracting the solution from the array TU

N - an integer giving the number of linear equations in (22)

T - a value of t at which the solution Y(t) is required

Y - the subroutine entry for obtaining estimated values of Y(t)

DH - the Jacobian matrix of the function H with respect to its arguments

DI - an integer giving the number of shooting points

DT - the integration interval for finding a value of the solution Y(t)

IL - an integer giving the shooting point immediately below T

IP - an integer pointing to that portion of the array TU containing the numbers of non-zero derivatives in each column of the Jacobian matrix of equation (22)

IU - an integer giving the shooting point immediately above T

IW - an integer pointing to that portion of the array TU used as work space by the routine NS03AD

IY - an integer pointing to that portion of the array TU used in routines DD03DD and DD03GD, to hold Y(t) and its derivatives with respect to $y_i(t_j)$ and μ_i, where t_j is some shooting point.

JK - an integer used for extracting the solution from the array TU

KN - an integer pointing to that portion of the array TU used in routines DD03DD and DD03GD, for accumulating mismatches at the matching points, and their derivatives with respect to the parameters μ_i

LP - the output stream number for error messages

LW - an integer giving the double words of work space available to NS03AD

MT - an integer pointing to that portion of the array TU used in routine DD03GD, to hold the Jacobian matrix of equation (22)

MU - the vector of eigenvalue parameters μ

MV - an integer pointing to that portion of the array TU used in routines DD03DD and DD03GD, for holding the results of integrating the variational equations across each shooting interval

NI - the integer n

NZ - the number of non-zero elements in the Jacobian matrix of equation (22)

PI - the integer p

PO - an integer equal to $n+p+1$

QI - an integer giving the number of shooting points actually used

QJ - an integer giving the maximum number of shooting points

TT - the shooting point used for obtaining the solution at a point T

TU - a work space array consisting of the following parts:

(i) 2 × QI elements giving the shooting and matching points;

(ii) n elements for use in subroutine DD03ED, for storage during numerical differentiation of the function G;

(iii) a sub-array MV (n, 2n, QI - 1) for use in subroutines DD03DD and DD03GD, for holding the results of integrating the variational equations across each shooting interval;

(iv) a sub-array KN (n, p + 1, QI - 1) for use in subroutines DD03DD and DD03GD, for accumulating mismatches at the matching points, and their derivatives with respect to the parameters μ_i ;

(v) a sub-array Y(n×(n+p+1)) for use in subroutines DD03DD and DD03GD, to hold Y(t) and its derivatives with respect to $y_i(t_j)$ and μ_i , where t_j is some shooting point;

(vi) 6n×(n+p+1) elements for use in subroutines DD03ID and DD03JD, for storage of immediate results during the Runge-Kutta integration process;

(vii) a sub-array YU (n× QI + p) used in subroutines DD03DD and DD03GD, and called X in subroutine NS03AD, used to hold the current values of y_i at the shooting points, and μ_i ;

(viii) an INTEGER*2 array IP(n× QI + p + 1) for use in subroutines DD03GD and NS03AD, to define the number of non-zero derivatives in each column of the Jacobian matrix of equation (22) ;

(ix) an INTEGER*2 array IRN((n ×QI + p) × (2n + p)) for use in the same routines to define the positions of the non-zero derivatives ;

(x) a work space array W for subroutine NS03AD, of which the first n×QI + p elements are also used by the name GA by subroutine DD03GD .

IGA - an integer pointing to that portion of the array TU used in routine DD03GD, to hold the right hand side of equation (22)

IRN - an integer pointing to that portion of the array TU containing the row numbers of the non-zero derivatives in the Jacobian matrix of equation (22)

ITU - the dimension of the work space array TU

IYU - an integer pointing to that portion of the array TU used in routines DD03DD, DD03GD and NS03AD, to hold the current values of y_i at the shooting points, and μ_i

SAC - error bound on the sum of squares of the residuals

ZET - a default value for ZETA if this is zero

EPSI - error bound for the elements of Y(t) during integration

FACT - the ratio of EPSI to ZETA

IAUX - an integer pointing to that portion of the array TU used in routine DD03ED, for storage during numerical differentiation of the function G

LAST - an integer pointing to the last of the non-zero derivatives of the Jacobian matrix of equation (22)

MAXF - a default value for MAXFUN

YRES - the solution Y(t) at a specified value T of t

ZERO - a constant equal to 0.0

ZETA - error bound on the root-mean-square of the residuals of the solution

DD03DD - the subroutine for performing a first integration to determine the shooting intervals, if these are not specified

DD03ED - an initialization entry to the subroutine called by the Runge-Kutta integration routine to calculate G, and if required its derivatives

DD03GD - the subroutine for performing the first integration if the shooting intervals are specified, and for determining the sparsity pattern of the Jacobian matrix of equation (22)

DD03HD - a side entry to DD03GD used for performing subsequent integrations during the iteration process

DD03JD - the subroutine for integrating across a single shooting interval, used here to obtain the solution Y(t) at specified points

DD03KD - a subroutine used for compacting the work space array TU, by shifting down the sub-arrays when the number of shooting points is known

DUMMY - a dummy argument supplied to subroutine NS03AD for options which are not used

NS03AD - the library subroutine used for solving the non-linear algebraic equations (16) and (17)

YMUST - increment in Y and μ for differentiation of the function G

LPRINT - the number of iterations between printouts

MAXFUN - the maximum number of integrations

The BLOCKDATA subroutine for common block DD03CD is used to provide certain default values for the program. The variables are as follows:

COMMON DD03CD:

ZET - a default value (10^{-13} for double precision working) for the error bound on the residuals of the solution

FACT - the ratio of EPSI to ZETA (default value 1.0)

MAXF - a default value (10) for the maximum number of integrations

LP - the output stream number (default value 6)

CFAC - the factor allowed for the growth of complementary solutions (default value 10.0)

The subroutine DD03DD is called by DD03AD to perform a first integration and determine the shooting intervals if they are not specified. In any case, by calls to the entry Y of subroutine SETY, it obtains initial estimates at the shooting points.

SUBROUTINE DD03DD:

A - the boundary point a

B - the boundary point b

J - an integer equal to $n+1$

K - an integer index used for various purposes

T - the value of t during integration

Y - the subroutine entry for obtaining initial estimates of Y(t)

DI - an integer giving the number of shooting points (set negative if no integration is performed to obtain them)

DT - the integration step, for use by the Runge-Kutta integration routine

II - an integer index used for various purposes

JJ - an integer index used for various purposes

JK - an integer used for counting the forward shooting intervals from a to b

KK - an integer used for counting the backward shooting intervals from b to a

KN - an array for accumulating the mismatches - Γ_{oi} at the matching points, and their derivatives N_{oi} with respect to the parameters μ_i

LI - an integer giving the maximum number of matching points that may be used

MO - an integer equal to 2n

MV - an array for holding the results M_{oi} of integrating the variational equations across each shooting interval

NI - the integer n

PO - an integer equal to $n+p+1$

PP - an integer equal to $p+1$

QI - an integer giving the maximum number of shooting points that may be used

TT - a variable used as a value of t or an interval in t for each shot

TU - an array for storing the shooting and matching points

YU - an array used to hold the initial estimates of y_i at the shooting points

AUX - a work space array to hold Y(t), its derivatives with respect to $y_i(t_j)$ and μ_i (where t_j is some shooting point) and intermediate results during the integration process

EPSI - error bound for the elements of Y(t) during integration

DDO3JD - the subroutine for integrating across a single shooting interval

The subroutine DDO3ED with entry DDO3FD is called by the Runge-Kutta integration routine, to calculate the function G, and if required its derivatives.

SUBROUTINE DDO3ED:

G - the subroutine defining the differential equations (5) or (9)

I - an integer index used for various purposes

J - an integer index used for various purposes

K - an integer index used for various purposes

L - an integer used for accessing the derivatives of G in the array VF

M - an integer used for accessing the differentials of Y(t) in the array Y

S - a storage location for elements of μ during numerical differentiation

T - a value of t at which the function G is to be calculated

Y - an array holding the value of Y(t) at which the function G is to be calculated, together with finite differentials of Y(t) for numerical differentiation

LO - an integer giving the number of differential equations being integrated, n if only equations (5) or (9) are being integrated, or $n(n+p+1)$ if the variational equations (14) are also being integrated

MU - the vector of eigenvalue parameters μ

NI - the integer n

PO - an integer equal to $n+p+1$

VF - an array for holding the value of the function G, together with its derivatives or differentials

AUX - a work space array of n elements for use during the calculation of derivatives or differentials of G

SUM - an accumulator for use during the matrix multiplication of the Jacobian matrix of G by the finite differentials of Y(t)

DUMMY - a dummy argument to the subroutine G when it does not calculate analytical derivatives

YMUST - increment in Y and μ for differentiation of the function G

The subroutine DD03GD with entry DD03HD is called by DD03AD to determine the sparsity pattern of the Jacobian matrix of equation (22). If shooting intervals are specified, it has to perform the first integration to do this. The side entry DD03HD is called by NS03AD to perform subsequent integrations, and the evaluation of the Jacobian matrix.

SUBROUTINE DD03GD:

H - the subroutine defining the boundary conditions (6), (8), (10) or (11)

I - an integer index used for various purposes

K - an integer index used for various purposes

M - an integer giving the number of linear equations in (22)

N - an integer giving the number of unknown corrections in equation (22) (M = N)

T - the value of t during integration

DH - the Jacobian matrix of the function H with respect to its arguments

DI - an integer giving the number of shooting points (negative on entry if no integration has yet been performed)

DT - the shooting interval for each single shot

GA - the vector of residuals or mismatches (the negative of the right hand side of equation (22))

II - an integer index used for various purposes

IP - an INTEGER*2 array defining the number of non-zero derivatives in each column of the Jacobian matrix of equation (22)

I1 - an integer giving the number of empty rows at the top of each column of the Jacobian matrix of equation (22)

JJ - an integer index used for various purposes

KK - an integer index used for various purposes

KN - an array for accumulating mismatches at the matching points, and their derivatives with respect to the parameters μ_i

KI - an integer pointing to the first non-zero term in each column of the Jacobian matrix of equation (22)

K2 - an integer pointing to the last non-zero term in each column of the Jacobian matrix of equation (22)

LI - an integer giving the number of matching points used

MO - an integer equal to 2n

MT - an array used for storing the Jacobian matrix of equation (22) in compressed form for use by NSO3AD

MU - the vector of eigenvalue parameters μ

MV - an array for holding the results of integrating the variational equations across each shooting interval

NI - the integer n

NO - an integer equal to $n+p$

NZ - the number of non-zero elements in the Jacobian matrix of equation (22)

PO - an integer equal to $n+p+1$

PP - an integer equal to $p+1$

QI - an integer giving the number of shooting points used

TT - the value of t at the end of a shooting interval, or matching point

TU - an array containing the shooting and matching points

YU - an array used to hold the current estimates of y_i at the shooting points, and the current estimates of μ_i

AUX - a work space array to hold Y(t), its derivatives with respect to $y_i(t_j)$ and μ_i (where t_j is some shooting point) and intermediate results during the integration process

IRN - an INTEGER*2 array defining the row numbers of the non-zero derivates in the Jacobian matrix of equation (22)

EPSI - error bound for the elements of Y(t) during integration

DDO3JD - the subroutine for integrating across a single shooting interval

NCALL - the number of times DDO3HD has been called, or number of integrations performed

The subroutine DDO3ID is the Runge-Kutta integration routine, which given Y(t), integrates over one step h, to find $Y(t+h)$, and also an estimate of the local truncation error. Besides these n equations, it integrates a further $\ell - n$ equations (the variational equations) which do not affect the derivatives of the first n variables.

SUBROUTINE DDO3ID:

E - an array used for storing intermediate values of Y(t) and the error in $Y(t+h)$

1. Purpose

This subroutine uses the method of multiple shooting to solve numerically a system of ordinary differential equations constituting a two-point boundary-value or eigenvalue problems and having the form

$$\frac{d}{dt}\underline{y}(t) = \underline{g}(t, \underline{y}(t), \underline{\mu}) \qquad .... (1)$$

$$\underline{h}(\underline{y}(a), \underline{y}(b), \underline{\mu}) = \underline{0} \qquad .... (2)$$

where $\underline{y}(t)$ is an n-vector of unknowns at values of the scalar t in the range $[a,b]$, $\underline{\mu}$ is a p-vector of unknown scalars, $\underline{g}$ is an n-vector function and $\underline{h}$ is an $(n + p)$ vector function. Equation (1) constitutes the differential equation itself and equation (2) its boundary conditions.

It should be noted that these problems are normally written as an equation of second or higher order, or a system of such equations. To use this subroutine they should be reduced to a system of two or more first-order equations.

In the case of an eigenvalue problem one or more of the parameters μ_i are the required eigenvalues and one or more of the equations (2) are usually normalising conditions. If an eigenvalue is complex it must be broken down into two real parameters μ_i .

The subroutine works by using NS03AD to improve iteratively approximations to $\underline{\mu}$ and to $\underline{y}$ at the "shooting-points" $a = t_1 < t_2 < .. < t_q = b$. Runge-Kutta integrations are performed forwards and backwards from each shooting point to "matching" points t_i', $i = 1,2, \ldots q - 1$, where $t_i \leqslant t_i' \leqslant t_{i+1}$ and the iteration aims for continuity at these points, as well as satisfaction of equation (2).

It is important that the problem be scaled so that all the variables y_i and μ_j are of similar size.

2. Argument lists

Main entry:

CALL DD03AD (A,B,N,P,G,H,DH,Y,U, ZETA, YMUST, MAXFUN, LPRINT,
Q, TU, ITU)

Entry for subsequent evaluation of solution $\underline{y}(t)$:

CALL DD03BD (T, YRES)

DD03AD arguments

A,B are REAL*8 variables to be set by the user to the values a,b of the variable t where the boundary conditions are imposed. It is necessary for the condition $a < b$ to hold.

N (INTEGER*4) is the number n of dependent variables y_i and of differential equations.

P (INTEGER*4) is the number of parameters μ_i.

G is the name of a subroutine with arguments (T,Y,U,FG,DG,N) which must be written by the user. When given values of t,y and μ in T,Y and U it must calculate the vector function $\underline{g}\ (t,\underline{y},\underline{\mu})$ and place it in the array FG. Optionally (see YMUST, below) it may in addition place the derivatives

$\frac{\partial g_i}{\partial y_j}$ in DG(i,j), i,j = 1.2, ..., n and the derivatives

$\frac{\partial g_i}{\partial \mu_j}$ in DG(i, j + n), i = 1,2, ... n, j = 1,2, ... p,

where DG is an array with dimensions (n,p + n). DD03AD passes n to the subroutine G.

H is the name of a subroutine with arguments (YA, YB, U, FH, DH, NP) which must be written by the user. When given values of $\underline{y}(a)$, $\underline{y}(b)$, and $\underline{\mu}$ in YA, YB and U it must, for i = 1,2, ..., n + p, place h_i in FH(i), $\frac{\partial h_i}{\partial y_j(a)}$ in DH(i,j), j = 1,2, ..., n, $\frac{\partial h_i}{\partial y_j(b)}$ in DH(i, n + j), j = 1,2, ..., n and $\frac{\partial h_i}{\partial \mu_j}$ in DH(i, 2n + j), j = 1,2, ..., p, where DH is the array of dimensions (n + p, 2n + p), which is the next argument of DD03AD. Any derivatives which are constants may be set in DH before DD03AD is called and do not then need to be set by H. NP is used to pass n + p to subroutine H in case this is required for a dummy dimension.

DH is a REAL*8 array of dimensions (n + p, 2n + p) whose function has just been described.

Y is the name of a subroutine with arguments (T,FY) which must be written by the user to specify a starting approximation to $\underline{y}(t)$. Given t it must return $\underline{y}(t)$ in FY.

U is a REAL*8 array of length p which must be set on entry to contain an estimate of $\underline{\mu}$ and on return it contains the best estimate of $\underline{\mu}$ found.

ZETA is a REAL*8 variable used to specify the accuracy required. Iteration continues until the root-mean-square discontinuity in the components of $\underline{y}$ at the matching points and the root-mean-square change to the components of $\underline{y}$ at the shooting points is less than ZETA. If ZETA is set non-positive it is replaced by a default value (see § 4). There is no facility for defining a relative accuracy, or for getting greater accuracy in particular variables, though appropriate scaling will help to achieve this end.

YMUST — is a REAL*8 variable which must be set to a step size for use in obtaining numerically (by one-sided differencing) the derivatives $\partial g_i/\partial y_j$ and $\partial g_i/\partial \mu_j$ in the case where G does not find them analytically. YMUST must be set to zero if G calculates these derivatives.

MAXFUN (INTEGER*4) — is the maximum number of integrations between a and b allowed during the iteration process. At least one integration is also performed before the iteration process starts. If MAXFUN is not positive it is replaced by a default value (see § 4).

LPRINT (INTEGER*4) — is used to control the printed output from NS03AD. If LPRINT = 0, no output is written to stream 6 apart from diagnostics. If LPRINT ≠ 0, output is written to stream 6 at the first and last iteration and at every IABS(LPRINT)th iteration in between, as described in the specification of NS03AD, LPRINT being the parameter IPRINT of that subroutine. If LPRINT > 0, a summary is output, giving some idea of the progress of the iteration. If LPRINT < 0, more details of the current approximate solution are output. The current solution X gives values of y_i at the shooting points, and values of μ_i. The current residual vector consists of the mismatches in y_i at the matching points, and the mismatches in the boundary conditions (2). The current vector V gives an approximation to the derivatives of ½ S with respect to the elements of the solution, where S is the sum of squares of the residuals.

Q (INTEGER*4) — must be set by the user to the number of shooting points he requires. If its value is less than 2 shooting points are chosen automatically. On exit it contains the number of shooting points used.

TU — is a REAL*8 work array of length ITU. If q shooting points are used the storage required is not more than $q(2 + 9\tfrac{1}{2}n + 10n^2 + 4\tfrac{1}{2}np) + 2 + 7n + 5n^2 + 9\tfrac{1}{2}p + 3\tfrac{1}{2}p^2 + 14np$ and about ⅔ of this may be sufficient. If Q is non-zero on entry than TU(i), i = 1,2, ... 2q - 1 must be set. Shooting takes place from the points TU(2i - 1), i = 1,2, ... q and can be in both directions. Matching takes place at the points TU(2i), i = 1,2, ..., q - 1 which should be distinct and interlaced between the shooting points (i.e. TU(j) ⩽ TU(j + 1), j = 1,2, ..., 2q - 2 and TU(2i) < TU(2i + 2), i = 1,2, ..., q - 2). After a successful entry the shooting and matching points used are stored in this way, TU(ITU - 1) holds the number of iterations taken. TU(ITU) is set to the number of words of TU actually needed or, if this is not known, to the upper bound given above. TU(ITU - 1) is also used to indicate error conditions (see § 3).

ITU (INTEGER*4)   must be set by the user to the length of the array TU.

DD03BD arguments

T   is a REAL*8 variable specifying a point at which $\underline{y}(t)$ is required.

YRES   is a REAL*8 array used for output of $\underline{y}(t)$.

3. Error returns

Error returns from DD03AD occur if the workspace TU is not large enough, if $a \geqslant b$, if NS03AD fails to solve the problem to the required accuracy or if more than MAXFUN iterations are needed. A message is output on stream 6 (unless LP is changed, see § 4) and the conditions may be recognised by TU(ITU - 1) being set to 0, -1, -2 or MAXFUN + 1, respectively. If DD03AD is entered with $Q > 0$ but the shooting and matching points in TU are unsuitable then a message is printed and execution continues as if Q had been zero.

4. Use of common

The subroutine contains a common block called DD03CD

```
COMMON/DD03CD/ZET,FACT,MAXF,LP,CFAC
```

These variables are set to 1D-13,1D0,10,6,1D1 by BLOCK DATA. ZET and MAXF give default values for ZETA and MAXFUN, and LP gives the stream number for diagnostic messages or zero if no such messages are required. FACT controls the Runge-Kutta integration; if ϵ = FACT* ZETA then the steps are adjusted so that the error in y_i across any shooting interval is less than ϵ if $\epsilon > 0$ or $|\epsilon|(|\epsilon| + y_i)$ if $\epsilon < 0$. CFAC controls the automatic choice of shooting and matching points; it should be increased for less points (but probably more convergence difficulties) and vice-versa.

5. Other Subroutines

DD03AD is in fact a package of subroutines. Besides DD03AD/BD/CD already mentioned it contains subroutines and entry points called DD03DD-KD. It calls Library Subroutines NS03AD, MA17AD,MC02AD,MC09AD and TD02AD.

6. Method

The subroutine uses an extension of the Multiple shooting Method described by M.R. Osborne (On Shooting Methods for Boundary Value Problems, J. Math. Anal. Appl. 27, 1969) and by H.B. Keller (Numerical Methods for Two-Point Boundary Value Problems, Blaisdell, 1968), using for the associated initial value problems a fourth order Runge-Kutta method described by R. England (Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations, Computer Journal, 12, 1969). It is hoped that a Culham report by R. England, describing the subroutine in detail, will be available by the end of 1973 (A program for the solution of boundary-value problems for systems of ordinary differential equations, CLM/PDN 3/73).

7. Example

For a very simple example consider the eigenvalue problem

$$\frac{d^2\varphi(t)}{dt^2} + \lambda\,\varphi(t) = 0$$

with boundary conditions

$$\frac{d\varphi}{dt}(0) = 0\,, \quad \varphi(\pi/2) = 0$$

and the additional boundary conditions

$$\varphi(0) = 1$$

fixes the normalisation. To reduce this problem to the required form write

$$y_1(t) = \varphi(t)\,, \quad y_2(t) = \frac{d\varphi(t)}{dt}$$

to give the equations

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -\lambda y_1$$

$$y_2(0) = 0$$

$$y_1(\pi/2) = 0$$

$$y_1(0) = 1$$

For a first approximation we take zero for λ and a straight line with correct end values for φ.

EXAMPLE PROGRAM

```
C    DD03 TEST
      IMPLICIT REAL*8(A-H,O-Z)
      EXTERNAL G,H,Y                       declare subroutines as external references
      DIMENSION TU(1000)                   workspace
      DIMENSION YRES(2)                    to contain $y_i(t)$ values for printing
      DIMENSION DH(3,5)                    to contain derivatives of boundary
                                             conditions
      INTEGER P,Q
      A=0.       )
      B=1.570796)                          range of integration
      N=2                                  no. of equations
      P=1                                  no. of parameters $\mu_i$
      DO 10 I-1,3)
      DO 10 J=1,5)
10    DH(I,J)=0. )
      DH(1,2)=1. )                         set derivatives of boundary conditions
      DH(2,3)=1. )
      DH(3,1)=1. )
      U=0.                                 estimate of $\lambda$
      ZETA=1E-4                            accuracy
      YMUST=0.                             to indicate derivatives supplied
                                             analytically
      MAXFUN=15                            max. no. of iterations
      LPRINT=1                             obtain printout at each NS03 iteration
      Q=0                                  shooting points to be chosen
                                             automatically
      ITU=1000                             size of workspace
      CALL DD03AD(A,B,N,P,G,H,DH,Y,U,
     1 ZETA,YMUST,MAXFUN,LPRINT,Q,TU,ITU)
      DO 15 I=1,11            )
      T=1.570796*(I-1)/10     )            print out solution at 11 equally
      CALL DD03BD(T,YRES)     )              spaced points
15    WRITE(6,20)T,YRES(1)    )
20    FORMAT(2F12.6)
      STOP
      END

      SUBROUTINE G(T,Y,U,FG,DG,N)          to evaluate g and its derivatives
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(2),FG(2),DG(N,3)
      FG(1)=Y(2)
      FG(2)=-U*Y(1)
      DG(1,1)=0.
      DF(2,1)=-U
      DG(1,2)=1.
      DG(2,2)=0.
      DG(1,3)=0.
      DG(2,3)=-Y(1)
      RETURN
      END
```

```
SUBROUTINE H(YA,YB,U,FH,DH,NP)              to evaluate boundary condition
                                              function h
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION YA(2),YB(2),FH(3),DH(NP,5)
FH(1)=YA(2)
FH(2)=YB(1)
FH(3)=YA(1)-1.
RETURN
END
```

```
SUBROUTINE Y(T,FY)                          to provide initial values of  y
IMPLICIT REAL*8(A-H,O-Z)                    using straight-line interpolation
DIMENSION FY(2)
FY(2)=-1./1.570796
FY(1)=1.+T*FY(2)
RETURN
END
```

October, 1973.

ACKNOWLEDGMENTS

I should particularly like to express my gratitude to two people who have made possible the publication of this report, after my own departure from Culham Laboratory to take up employment in the Dirección General de Planeación Educativa of the Secretaria de Educación Pública in Mexico City.

Dr J.K. Reid has made considerable finishing touches and improvements to the subroutine package DD03AD, in order to include it in the Harwell Subroutine Library.

Mr P.M. Keeping has supervised the typing and reproduction of this report at Culham, and has also adapted those routines which were necessary for the program to be used at Culham.

I would add my thanks to Mrs Olive Thorne who has so capably typed the report from my manuscripts returned from Mexico.

REFERENCES

1. England, R. - Error estimates for Runge-Kutta type solutions to systems of ordinary differential equations, Computer Journal, 12, 1969.

2. Fletcher, R. - A modified Marquardt subroutine for non-linear least squares, AERE Harwell Report R6799, H.M. Stationery Office, 1971.

3. Keller, H.B. - Numerical Methods for Two-point Boundary Value Problems, Blaisdell, 1968.

4. Lance, G.N. and Rogers, M.H. - The symmetric flow of a viscous fluid between two infinite rotating discs, Proc. Roy. Soc. 266, (1962), pp. 109 - 121.

5. Lashmore-Davies, C.N.- Stabilization of a low-density plasma in a simple magnetic mirror by feed back control, J.Phys. A: Gen.Phys. 4, 1971.

6. Marquardt, D.W. - An algorithm for least squares estimation of non-linear parameters, J. SIAM, 11, 1963.

7. Osborne, M.R. - On Shooting Methods for Boundary Value Problems, J. Math. Anal. Appl. 27, 1969.

8. Reid, J.K. - Fortran Subroutines for the Solution of sparse systems of non-linear equations, AERE Harwell Report R7293, H.M. Stationery Office, 1972.

APPENDIX A. PROGRAM LISTINGS

```
      SUBROUTINE SETDEF(NI,NO,NN,ITU,QI,DH,TU,R,YR)
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER NI,NO,NN,ITU,QI,PI,RI,SI,DI,MAXFUN,LPRINT,L,I,C
      DIMENSION DH(NO,NN),TU(ITU),R(QI),YR(NI,QI)
      COMMON/HDATA/C,N
      COMMON/DD03CD/ZET,FACT,MAXF,LP
      EXTERNAL G,H,Y
      DATA A,B/0D0,1D0/,ZETA/1D-5/,YMUST/0D0/,EPSI/-1D-4/
      DATA RI,SI/0,0/,MAXFUN,LPRINT/10,0/,DI/0/,M/5/
      NAMELIST/VALUES/RI,SI,A,B,ZETA,YMUST,EPSI,MAXFUN,LPRINT,DI,LP
      DO 101 K=1,NN
      DO 101 J=1,NO
  101 DH(J,K)=0.0
      N=NI
      C=NO
      READ(M,VALUES)
      L=LP
      PI=NO-NI
      GO TO 200
      ENTRY PROGC(S,MU)
      REAL*8S(1),MU(1)
      IF(RI.GT.0 .OR. PI.LT.1)GO TO 113
      DO 102 K=1,PI
  102 MU(K)=0.
  113 CONTINUE
      WRITE(L,103)NI,NO,NN,ITU,PI,QI,RI,SI,A,B
      DO 104 K=1,NN
  104 WRITE(L,105)K,(DH(J,K),J=1,NO)
      IF (SI.GT.0) WRITE(L,106) (S(J),J=1,SI)
      IF (RI.EQ.0) GO TO 107
      K=IABS(RI)
      DO 108 J=1,K
  108 WRITE(L,109)R(J),(YR(I,J),I=1,NI)
      IF (PI.GT.0.AND.RI.GT.0) WRITE(L,110) (MU(J),J=1,PI)
  107 WRITE(L,111)LPRINT,MAXFUN,       YMUST,ZETA,EPSI
      IF (DI.GT.0) WRITE(L,112)DI,TU(1),(TU(2*J-2),TU(2*J-1),J=2,DI)
      IF(ZETA*EPSI.NE.0.)FACT=EPSI/ZETA
      CALL SETY(A,B,RI,NI,R,YR)
      CALL DD03AD(A,B,NI,    PI,G,H,DH,Y,MU,ZETA,YMUST,MAXFUN,LPRINT,DI,
     1 TU,ITU)
      IF(TU(ITU-1).EQ.0.)GO TO 100
      IF(SI.LT.1)GO TO 60
      DO 50 J=1,SI
   50 R(J)=S(J)
      RI=SI
      GO TO 80
   60 DO 70 J=1,DI
   70 R(J)=TU(2*J-1)
      RI=DI
   80 DO  90 J=1,RI
   90 CALL DD03BD(R(J),YR(1,J))
  100 DO 114 J=1,RI
  114 WRITE(L,115)R(J),(YR(I,J),I=1,NI)
      IF (PI.GT.0) WRITE(L,116) (MU(J),J=1,PI)
      K=DI+DI-1
      IF(DI.GT.0)WRITE(L,118)DI,(TU(J),J=1,K)
      J=TU(ITU)
      WRITE(L,117)J
  200 RETURN
  103 FORMAT(1H156X35HNUMBER OF DEPENDENT VARIABLES Y: N=I4/
     1        1H053X38HNUMBER OF BOUNDARY CONDITIONS: N+P=NO=I4/
     2        1H027X64HNUMBER OF VARIABLES INVOLVED IN BOUNDARY CONDITION
     3S: 2N+P+1= NN=I4/1H045X44HNUMBER OF WORDS OF WORK SPACE PROVIDED:
     4ITU=I6/   1H061X30HDIMENSION OF EIGENVALUE MU: P=I4/
     5        1H029X62HMAXIMUM STORAGE FOR TABULATION POINTS OR ESTIMATIO
     6N POINTS: R=I4/
     7        1H025X66HNUMBER OF ESTIMATED VALUES OF Y (POSITIVE IF MU IS
     8 ESTIMATED): RI=I4/1H049X42HNUMBER OF SPECIFIED TABULATION POINTS:
     9 SI=I4/
     A        1H036X55HVALUES OF INDEPENDENT VARIABLE T AT BOUNDARY POINT
     BS: A=F9.4/90X2HB=F9.4/1H032X37HDEFAULT VALUES FOR JACOBIAN MATRIX
     CDH/26X50HOR COEFFICIENTS OF BOUNDARY CONDITIONS H1,H2,H3,H0/1X)
  105 FORMAT(1XI4,12(1XF9.4)/(5X12(1XF9.4)))
  106 FORMAT(28H0SPECIFIED TABULATION POINTS/1X/(6XF9.4))
```

```
109   FORMAT(23HOESTIMATED VALUES OF Y(F11.6,1H)1P8E12.4/(35X8E12.4))
110   FORMAT(1H034X32HESTIMATED VALUE OF EIGENVALUE MU//(11X1P10E12.4))
111   FORMAT(1H044X47HNUMBER OF ITERATIONS BETWEEN PRINTOUTS: LPRINT=I4/
     /1H044X47HMAXIMUM NUMBER OF INTEGRATIONS ALLOWED: MAXFUN=I4//25X67
     7 HINCREMENT IN Y AND MU FOR DIFFERENTIATION OF THE FUNCTION G: YMU
     1ST=      1PE10.2/1H061X30HERROR BOUND ON SOLUTION: ZETA=1PE10.2
     2    /1H024X67HERROR BOUND FOR THE DEPENDENT VARIABLES Y DURING INT
     3EGRATION: EPSI=1PE10.2)
112   FORMAT(1H0I2,26H SPECIFIED SHOOTING POINTS/1X/(5X2(1XF9.4)))
115   FORMAT(23H0TABULATED VALUES OF Y(F11.6,1H)1P8E12.4/(35X8E12.4))
116   FORMAT(1H034X32HTABULATED VALUE OF EIGENVALUE MU//(11X1P10E12.4))
117   FORMAT(1H0I6,24H WORDS OF WORKSPACE USED)
118   FORMAT(1H0I2,23H ACTUAL SHOOTING POINTS//(5X2(1XF9.4)))
         END


      SUBROUTINE SETY(A,B,RI,NI,R,YR)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION R(1),YR(NI,1)
      INTEGER RI
      RETURN
      ENTRY Y(T,FY)
      DIMENSION FY(1)
      IF(RI.NE.0)GO TO 20
      DO 10 JJ=1,NI
10    FY(JJ)=0.
      RETURN
20    II=IABS(RI)
30    II=II-1
      IF((R(II)-T)        .GT.0.)GO TO 30
      U=R(II+1)-R(II)
      V=(T-R(II))/U
      U=(R(II+1)-T)/U
      DO 40 JJ=1,NI
40    FY(JJ)=V*YR(JJ,II+1)+U*YR(JJ,II)
      RETURN
      END


      SUBROUTINE H(YA,YB,MU,FH,DH,NP)
      IMPLICIT REAL*8(A-H,O-Z)
      REAL*8 YA(1),YB(1),MU(1),FH(1),DH(1)
      COMMON/HDATA/C,N
      INTEGER C,N,Q,S,PI,QH,RH,SH
      PI=C-N
      QH=C*(N+C)
      DO 1 Q=1,C
      QH=QH+1
      FH(Q)=-DH(QH)
      RH=Q
      SH=Q+C*N
      DO 2 S=1,N
      FH(Q)=FH(Q)+DH(RH)*YA(S)+DH(SH)*YB(S)
      RH=RH+C
2     SH=SH+C
      IF (PI.LT.1) GO TO 1
      DO 3 S=1,PI
      FH(Q)=FH(Q)+DH(SH)*MU(S)
3     SH=SH+C
1     CONTINUE

      RETURN
      END
      SUBROUTINE DD03AD(A,B,NI,PI,G,H,DH,Y,MU,ZETA,YMUST,MAXFUN,LPRINT,
     1DI,TU,ITU)
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON/NS03HD/ITEST
      COMMON/DD03CD/ZET,FACT,MAXF,LP,CFAC
      INTEGER QJ,PO,DI,PI,QI
      REAL*8 MU(PI),TU(ITU),ZERO/0D0/,DH(1)
      EXTERNAL DD03HD,G,H,Y
      JTEST=ITEST
      ITEST=1
      IF(A.GE.B) GO TO 360
C IF SHOOTING POINTS ARE GIVEN THEN CHECK THEM.
      IF(DI.LE.1)GO TO 8
      J=2*DI-2
      DO 3 I=1,J
3     IF(TU(I).GT.TU(I+1))DI=0
      J=J-2
      DO 5 I=2,J,2
5     IF(TU(I).GE.TU(I+2))DI=0
      IF(DI.EQ.0 .AND. LP.GT.0)WRITE(LP,7)
7     FORMAT(' DD03 HAS BEEN GIVEN FAULTY SHOOTING OR MATCHING POINTS.
     1EXECUTION WILL CONTINUE AS IF Q=0')
8     IF(DI.LE.1)DI=0
C IF NOT SET, PROVIDE DEFAULT VALUES FOR ZETA AND MAXFUN.
      IF (ZETA.LE.ZERO)ZETA=ZET
      IF (MAXFUN.LE.0) MAXFUN=MAXF
      EPSI=ZETA*FACT
      PO=NI+PI+1
C ALLOCATE SPACE IN TU ON BASIS OF GIVEN NUMBER OF SHOOTING POINTS OR
C GREATEST NUMBER THAT ALLOWS DD03E/D TO BE EXECUTED.
9     QJ=DI
      IF(QJ.LE.1)QJ=1+(ITU-2-NI-7*NI*PO-PI-NI)/(2+2*NI*NI+NI*(PI+2))
      TU(ITU)=QJ*(2+10*NI*PO-NI*(1+11*PI)/2)+PO*(2+5*NI)+9*NI*PI+PI*(15
     1 +7*PI)/2
      IF(QJ.LE.1)GO TO 320
      IAUX=2*QJ+1
      MV=IAUX+NI
      KN=MV+2*NI*NI*(QJ-1)
      IY=KN+NI*(PI+1)*(QJ-1)
      IYU=IY+7*NI*PO
      IP=IYU+NI*QJ+PI
      IF(IP.LE.ITU)GO TO 20
      DI=0
      IF(LP.GT.0)WRITE(LP,10)
10    FORMAT(' DD03 HAS BEEN GIVEN INSUFFICIENT WORKSPACE FOR THE ',
     1'SPECIFIED NUMBER OF SHOOTING POINTS. EXECUTION TO CONTINUE AS IF
     2Q=0')
      GO TO 9
C INITIALIZE SUBROUTINE CALLED BY RUNGE-KUTTA INTEGRATION ROUTINE.
20    CALL DD03ED(NI,YMUST,G,MU,TU(IAUX))
C FIND FIRST APPROX. SOLUTION AND, IF REQUIRED, FIND SHOOTING AND
C MATCHING POINTS.
      CALL DD03DD(A,B,NI,2*NI,PI+1,PO,QJ,DI,EPSI,TU,Y,
     1TU(IYU),TU(MV),TU(KN),TU(IY))
C JUMP IF DD03D WANTED MORE SPACE THAN IT WAS GIVEN.
      IF (DI.EQ.0) GO TO 320
      QI=IABS(DI)
      N=NI*QI+PI
C SHIFT ARRAYS IN WORKSPACE TO CORRESPOND WITH ACTUAL NUMBER OF
C SHOOTING POINTS.
      IF(QI.EQ.QJ)GO TO 210
      IAUX=2*QI+1
      CALL DD03ED(NI,YMUST,G,MU,TU(IAUX))
      I=MV
      J=2*NI*NI*(QI-1)
      MV=IAUX+NI
      CALL DD03KD(TU(MV),TU(I),J)
      I=KN
      KN=MV+J
      J=NI*(PI+1)*(QI-1)
      CALL DD03KD(TU(KN),TU(I),J)
      I=IY
      IY=KN+J
      J=7*NI*PO
      CALL DD03KD(TU(IY),TU(I),J)
      I=IYU
      IYU=IY+J
      CALL DD03KD(TU(IYU),TU(I),N)
      IP=IYU+N
210   IRN=IP+(N+2)/2
      NZ=(2*NI+PI)*N
      IGA=IRN+(NZ+1)/2
      MT=IGA+N
      LAST=MT+NZ
      IF(LAST.GT.ITU)GO TO 320
C FIND INTEGER ARRAYS IP AND IRN AND NUMBER (NZ) OF NON-ZEROS.
      CALL DD03GD(NI,2*NI,PO-1,PI+1,PO,DI,DH,EPSI,TU,MU,TU(IYU),TU(MV),
     1TU(KN),TU(IGA),NZ,TU(MT),TU(IRN),TU(IP),TU(IY),H)
       IW=IRN+(NZ+1)/2
      LW=ITU-IW+1
      SAC=ZETA*ZETA*N
C CALL NON-LINEAR EQUATION SOLVER.
      CALL NS03AD(DD03HD,N,N,TU(IYU),SAC, SAC,MAXFUN,LPRINT,TU(IW),
     1LW,TU(IRN),TU(IP),DUMMY,DUMMY,0,ZERO)
C TEST FOR NS03 ERROR FLAGS.
      TU(ITU)=DMAX1(LAST+ZERO,IW+TU(ITU))
      IF(TU(ITU-1).EQ.0)GO TO 320
      IF(TU(ITU-1).GT.MAXFUN)GO TO 340
      CALL MC02AD(TU(IW),TU(IW),S,N)
      IF(S.GT.SAC)GO TO 380
212   ITEST=JTEST
      RETURN
C
      ENTRY DD03BD(T,YRES)
      DIMENSION YRES(1)
C FIND THE INTERVAL BETWEEN SHOOTING POINTS IN WHICH T LIES.
      IL=0
      IU=DI+1
215   I=(IL+IU)/2
      IF((TU(2*I-1)-T)*(B-A))220,250,230
220   IL=I
      GO TO 235
230   IU=I
235   IF(IU.NE.IL+1)GO TO 215
C DECIDE WHETHER FORWARD OR BACKWARD SHOOTING IS APPROPRIATE AND PERFORM
C SHOOT.
      IF(IL.EQ.0)IL=1
      IF(IU.GT.DI)GO TO 240
C*    IF(TU(IL*2).LT.T)IL=IU
      IF(TU(2*IL).LT.T)IL=IU
240   I=IL
250   JK=IYU+(I-1)*NI-1
      TT=TU(2*I-1)
      DT=T-TT
      DO 260 I=1,NI
      K=I+JK
260   YRES(I)=TU(K)
      IF(DT.NE.ZERO)CALL DD03JD(.TRUE.,NI,NI,TT,DT,T,YRES,EPSI,TU(IY))
      RETURN
C
C ERROR RETURNS.
300   IF(LP.GT.0)WRITE(LP,310)
310   FORMAT('+ERROR RETURN FROM DD03 BECAUSE')
      GO TO 212
320   IF(LP.GT.0)WRITE(LP,330)
330   FORMAT(32X,' DIMENSION OF TU IS TOO SMALL')
      TU(ITU-1)=0
      GO TO 300
340   IF(LP.GT.0)WRITE(LP,350)
350   FORMAT(32X,'TOO MANY ITERATIONS NEEDED')
      GO TO 300
```

```
 360    IF(LP.GT.0)WRITE(LP,370)
 370    FORMAT(32X,'A.GE.B')
        TU(ITU-1)=-1
        GO TO 300
 380    IF(LP.GT.0)WRITE(LP,390)
 390    FORMAT(32X,'NS03 HAS FAILED TO SOLVE PROBLEM')
        TU(ITU-1)=-2
        GO TO 300
        END
        BLOCK DATA
        IMPLICIT REAL*8(A-H,O-Z)
        COMMON/DD03CD/ZET,FACT,MAXF,LP,CFAC
        DATA CFAC/1D1/
        DATA ZET,FACT,MAXF,LP/1D-13,1D0,10,6/
        END


        SUBROUTINE DD03KD(A,B,N)
        IMPLICIT REAL*8(A-H,O-Z)
        DIMENSION A(N),B(N)
        DO 10 I=1,N
 10     A(I)=B(I)
        RETURN
        END


        SUBROUTINE DD03JD(BN,LO,NI,T,DT,TT,Y,EPSI,AUX)
C INTEGRATES OVER A SINGLE INTERVAL.
        IMPLICIT REAL*8(A-H,O-Z)
        LOGICAL BN
        EXTERNAL DD03FD
        DIMENSION Y(LO),AUX(LO,6)
        COMMON/DD03CD/ZET,FACT,MAXF,LP,CFAC
C STORE INITIAL VALUE OF Y AND SHOOTING INTERVAL SIZE.
        DO 101 JJ=1,LO
 101    AUX(JJ,1)=Y(JJ)
        W=TT-T
C REDUCE STEP SIZE IF ONE INTEGRATION STEP WILL EXCEED MATCHING POINT.
        IF (W/DT.LT.1.0) DT=W
        V=T
 106    CALL DD03ID(LO,DD03FD,DT,V,Y,AUX(1,2),NI,AUX(1,3))
C DETERMINE NORM OF INTEGRATION ERROR IN PROPORTION TO EPSI, THEN
C DIVIDE BY FRACTION OF SHOOTING INTERVAL TO GIVE U.
        U=0.0
        DO 102 II=1,NI
        AUX(II,2)=DABS(AUX(II,2))/EPSI
        IF (EPSI.LE.0.0) AUX(II,2)=AUX(II,2)/(EPSI-DABS(Y(II)))
        IF (U.LT.AUX(II,2)) U=AUX(II,2)
 102    CONTINUE
        U=U*W/DT
C ADJUST STEP-SIZE ACCORDING TO SIZE OF U.
        IF (U.GE.0.25) GO TO 103
        DT=DT*3.0
        IF (U.GT.0.0081) DT=DT*0.3/DSQRT(DSQRT(U))
        GO TO 104
 103    IF (U.LT.0.75.OR.W/DT.GT.1E3.OR.V.EQ.V+DT/3.0) GO TO 104
        DT=DT/3.0
        IF (U.LT.33.1776) DT=DT*2.4/DSQRT(DSQRT(U))
        IF (U.LE.1.0) GO TO 104
C RESTORE INITIAL VALUES OF Y AND T.
        DO 105 II=1,LO
 105    Y(II)=AUX(II,1)
        V=T
        GO TO 106
C STORE NEW VALUES OF Y AND T.
 104    DO 107 II=1,LO
 107    AUX(II,1)=Y(II)
        T=V
C JUMP IF SHOOTING INTERVALS HAVE BEEN DETERMINED.
        IF (BN) GO TO 108
C TEST WHETHER NORM OF PROPAGATION MATRIX IS GREATER THAN 10 AND IF SO
C RETURN.
```

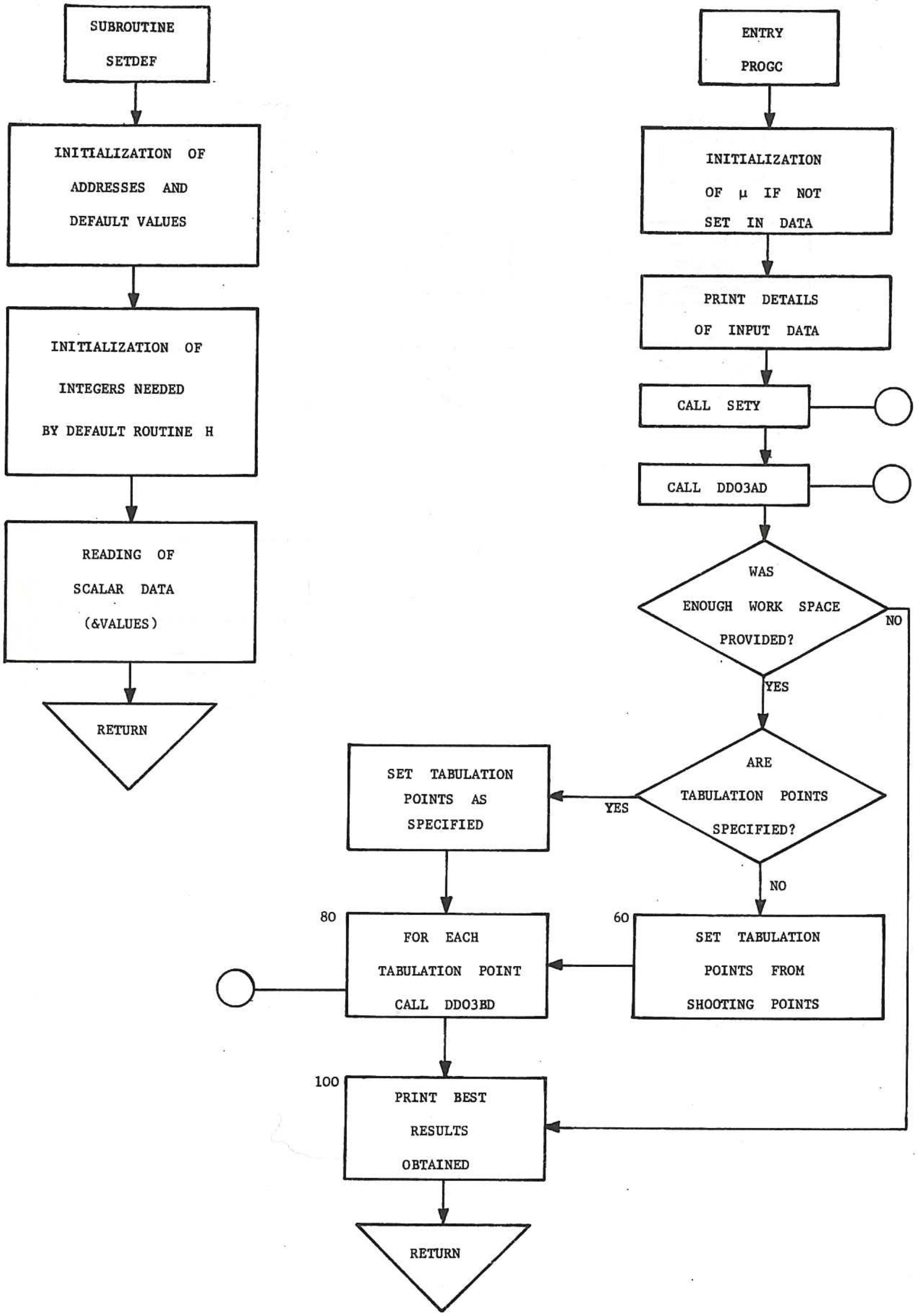
```
      DO 109 II=1,NI
      JJ=II+NI
      U=0.0
      DO 110 K=JJ,LO,NI
  110 U=U+DABS(Y(K))
      IF (U.GT.CFAC) GO TO 111
  109 CONTINUE
C JUMP IF NEXT INTEGRATION POINT WILL NOT EXCEED MATCHING POINT.
  108 IF ((TT-T)/DT.GT.1.0) GO TO 106
      DT=TT-T
C REDUCE STEP-SIZE AND JUMP IF STEP-SIZE WILL MAKE A SIGNIFICANT CHANGE
C IN T.
      IF (T.NE.T+DT) GO TO 106
  111 RETURN
      END

      SUBROUTINE DD03GD(NI,MO,NO,PP,PO,DI,DH,EPSI,TU,MU,YU,MV,KN,GA,
     1  NZ,MT,IRN,IP,AUX,H)
C THIS ENTRY (FORMERLY PACDIF) PREFORMS SHOOTING AND MATCHING IF
C THIS HAS NOT JUST BEEN DONE BY DD03D AND IN ANY CASE CALCULATES THE
C INTEGER ARRAYS IP AND IRN.
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER*2 IRN,IP
      INTEGER QI,MO,NO,PP,PO,DI
      REAL*8 TU(2,1),MU(PP),YU(1),MV(NI,MO,1),KN(NI,PP,1),GA(1),MT(1)
      DIMENSION DH(NO,MO),IRN(1),IP(1),AUX(NI,PO,7)
      NCALL=0
      DI=-DI
C RECORD THE NUMBER OF SHOOTING AND MATCHING POINTS.
      QI=IABS(DI)
      LI=QI-1
      IF (PP.LT.2) GO TO 103
C IF THERE IS A VECTOR MU ADD IT TO THE END OF THE VECTOR OF ESTIMATES.
      II=NI*QI
      DO 101 JJ=2,PP
      II=II+1
  101 YU(II)=MU(JJ-1)
      GO TO 103
      ENTRY DD03HD(N,YU,GA,H,MT)
C THIS ENTRY (FORMERLY MNYSHT) IS CALLED BY NS03 AND CALCULATES THE
C RESIDUALS AND THE JACOBIAN MATRIX.
      NCALL=NCALL+1
C IF THERE IS A VECTOR MU INITIALIZE IT FROM THE END OF TNE VECTOR OF
C ESTIMATES.
      IF (PP.LT.2) GO TO 103
      II=NI*QI
      DO 104 JJ=2,PP
      II=II+1
  104 MU(JJ-1)=YU(II)
  103 KK=NI*LI
      II=NI*QI
      CALL H(YU,YU(KK+1),YU(II+1),GA(KK+1),DH,NO)
C JUMP IF AN INTEGRATION HAS JUST BEEN PERFORMED BY DD03D OR THIS IS
C THE FIRST CALL OF DD03H.
      IF (DI.LE.0) GO TO 102
      IF(NCALL.EQ.1)GO TO 102
      I=0
      DO 105 KK=1,LI
C INITIALIZE PARTIAL DERIVATIVE ARRAY.
      DO 106 JJ=1,NI
      I=I+1
      DO 107 II=2,PO
  107 AUX(JJ,II,1)=0.0
      AUX(JJ,JJ+1,1)=1.0
  106 AUX(JJ,1,1)=YU(I)
```

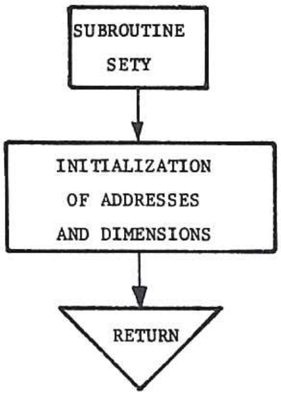
```
C SET UP FORWARD SHOOTING INTERVAL.
      T=TU(1,KK)
      TT=TU(2,KK)
      DT=TT-T
      IF (T.NE.TT)
     1 CALL DD03JD(.TRUE.,NI*PO,NI,T,DT,TT,AUX,EPSI,AUX(1,1,2))
C ADD RESULTS TO PROPAGATION MATRICES AND REINITIALIZE PARTIAL
C DERIVATIVE ARRAY.
      DO 109 JJ=1,NI
      I=I+1
      KN(JJ,PP,KK)=AUX(JJ,1,1)
      IF (PP.LT.2) GO TO 110
      DO 111 II=2,PP
      K=NI+II
      KN(JJ,II-1,KK)=AUX(JJ,K,1)
 111  AUX(JJ,K,1)=0.0
 110  DO 112 II=1,NI
      MV(JJ,II,KK)=AUX(JJ,II+1,1)
 112  AUX(JJ,II+1,1)=0.0
      AUX(JJ,JJ+1,1)=-1.0
 109  AUX(JJ,1,1)=YU(I)
      I=I-NI
C SET UP BACKWARD SHOOTING INTERVAL.
      T=TU(1,KK+1)
      TT=TU(2,KK)
      DT=TT-T
      IF(T.NE.TT)
     1 CALL DD03JD(.TRUE.,NI*PO,NI,T,DT,TT,AUX,EPSI,AUX(1,1,2))
C SUBTRACT RESULTS FROM PROPAGATION MATRICES.
      DO 105 JJ=1,NI
      IF (PP.LT.2) GO TO 115
      DO 116 II=2,PP
      K=NI+II
 116  KN(JJ,II-1,KK)=KN(JJ,II-1,KK)-AUX(JJ,K,1)
 115  K=NI
      DO 117 II=1,NI
      K=K+1
 117  MV(JJ,K,KK)=AUX(JJ,II+1,1)
 105  KN(JJ,PP,KK)=KN(JJ,PP,KK)-AUX(JJ,1,1)
C RESET MARKER IF INTEGRATION PERFORMED BY DD03B (ONEPAS).
102   DI=QI
      IF(NCALL.GT.0)GO TO 215
C
C FIND SPARSITY PATTERN OF JACOBIAN.
      K=1
      KK=NI*LI
      DO 119 II=1,NI
C SET POINTER ARRAYS FOR FIRST PROPAGATION MATRIX
      IP(II)=K
      DO 120 JJ=1,NI
      IF(JJ.NE.II .AND. TU( 1,1).EQ.TU( 2,1))GO TO 120
      IRN(K)=JJ
      K=K+1
 120  CONTINUE
C SET POINTER ARRAYS FOR DH/DY(A)
      DO 119 JJ=1,NO
      IRN(K)=KK+JJ
      K=K+1
 119  CONTINUE
      IF (LI.LT.2) GO TO 121
C SET POINTER ARRAYS FOR PAIRS OF BACKWARD AND FORWARD PROPAGATION
C MATRICES.
      DO 122 KK=2,LI
      DO 122 II=1,NI
      I=NI*(KK-1)+II
      IP(I)=K
      I=NI+II
      DO 123 JJ=1,NI
      IF(JJ.NE.II .AND. TU(1,KK).EQ.TU(2,KK-1))GO TO 123
      IRN(K)=NI*(KK-2)+JJ
      K=K+1
 123  CONTINUE
      I=NI*(KK-1)
      DO 122 JJ=1,NI
      IF(JJ.NE.II .AND. TU(1,KK).EQ.TU(2,KK))GO TO 122
      IRN(K)=I+JJ
      K=K+1
 122  CONTINUE
 121  KK=NI*LI
      DO 124 II=1,NI
C SET POINTER ARRAYS FOR LAST PROPAGATION MATRIX.
      I=KK+II
      IP(I)=K
      I=NI+II
      DO 125 JJ=1,NI
      IF(JJ.NE.II .AND. TU(1,QI).EQ.TU(2,LI))GO TO 125
      IRN(K)=KK-NI+JJ
      K=K+1
 125  CONTINUE
C SET POINTER ARRAYS FOR DH/DY(B)
      DO 124 JJ=1,NO
      IRN(K)=KK+JJ
      K=K+1
 124  CONTINUE
      IF (PP.LT.2) GO TO 129
C SET POINTER ARRAYS FOR DERIVATIVES WITH RESPECT TO MU.
      DO 126 II=2,PP
      I=NI*QI+II
      IP(I-1)=K
      I=1
      DO 127 KK=1,LI
      DO 127 JJ=1,NI
      IRN(K)=I
      K=K+1
 127  I=I+1
      I=MO+II
      KK=NI*LI+1
      DO 126 JJ=1,NO
      IRN(K)=KK+JJ-1
      K=K+1
 126  CONTINUE
 129  I=NI*QI+PP
      IP(I)=K
      NZ=K-1
      RETURN
C
C TRANSFER RESIDUALS INTO MATRIX GA.
215   II=1
      DO 220 KK=1,LI
      DO 220 JJ=1,NI
      GA(II)=KN(JJ,PP,KK)
220   II=II+1
      KK=NI*LI
C TRANSFER FIRST PROPAGATION MATRIX AND DH/DY(A) TO MT
      DO 240 II=1,NI
      K1=IP(II)
      K2=IP(II+1)-1
      DO 240 K=K1,K2
      JJ=IRN(K)
      IF(JJ.GT.NI)GO TO 230
      MT(K)=MV(JJ,II,1)
      GO TO 240
230   JJ=JJ-KK
      MT(K)=DH(JJ,II)
240   CONTINUE
      IF (LI.LT.2) GO TO 270
C TRANSFER PAIRS OF BACKWARD AND FORWARD PROPAGATION MATRICES TO MT.
      DO 260 KK=2,LI
      DO 260 II=1,NI
      I=NI*(KK-1)+II
      I1=NI*(KK-2)
      K1=IP(I)
      K2=IP(I+1)-1
      I=NI+II
      DO 260 K=K1,K2
      JJ=IRN(K)-I1
      IF(JJ.GT.NI)GO TO 250
      MT(K)=MV(JJ,I,KK-1)
      GO TO 260
250   JJ=JJ-NI
      MT(K)=MV(JJ,II,KK)
260   CONTINUE
270   KK=NI*LI
C TRANSFER LAST PROPAGATION MATRIX AND DH/DY(B) TO MT
      DO 290 II=1,NI
      I=KK+II
      K1=IP(I)
      K2=IP(I+1)-1
      I=NI+II
      DO 290 K=K1,K2
      JJ=IRN(K)-KK+NI
      IF(JJ.GT.NI)GO TO 280
      MT(K)=MV(JJ,I,LI)
      GO TO 290
280   JJ=JJ-NI
      MT(K)=DH(JJ,I)
290   CONTINUE
      IF (PP.LT.2) GO TO 320
C TRANSFER DERIVATIVES WITH RESPECT TO MU TO MT.
      DO 310 II=2,PP
      I=NI*QI+II
      K1=IP(I-1)
      K2=IP(I)-1
      I=MO+II
      DO 310 K=K1,K2
      KK=(IRN(K)-1)/NI+1
      JJ=IRN(K)-(KK-1)*NI
      IF(KK.GT.LI)GO TO 300
      MT(K)=KN(JJ,II-1,KK)
      GO TO 310
300   JJ=IRN(K)-NI*LI
      MT(K)=DH(JJ,I-1)
310   CONTINUE
320   RETURN
      END

      SUBROUTINE DD03ID(L,FUNC,H,T,Y,E,M,AUX)
C THIS SUBROUTINE IMPLEMENTS THE USUAL FOURTH ORDER RUNGE-KUTTA METHOD
C FOR L DIFFERENTIAL EQUATIONS AND USES ENGLAND'S (COMP. J. 12 (1969),
C P.166) ERROR ESTIMATE (6) FOR THE FIRST M VARIABLES. IT ASSUMES THAT
C THE FIRST M DERIVATIVES DO NOT DEPEND ON THE REMAINING VARIABLES.
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION Y(L),E(L),AUX(1)
C AUX IS A WORK ARRAY OF DIMENSION L*4
C Y(T) IS OVERWRITTEN BY Y(T+H),T BY T+H AND ERROR ESTIMATES ARE PLACED
C IN E.
      L2=L*2
      L3=L*3
      CALL FUNC(T,Y,AUX,L)
      F0=H/2.
      DO 10 I=1,L
10    E(I)=Y(I)+AUX(I)*F0
      CALL FUNC(T+F0,E,AUX(L+1),L)
      F0=H/4.
      DO 20 I=1,L
      I2=I+L
20    E(I)=Y(I)+(AUX(I)+AUX(I2 ))*F0
      CALL FUNC(T+H/2.,E,AUX(L2+1),L)
      F1=-H
      F2=H+H
      DO 30 I=1,L
      I2=I+L
      I3=I+L2
30    E(I)=Y(I)+F1*AUX(I2 )+F2*AUX(I3)
      CALL FUNC(T+H,E,AUX(L3+1),L)
      DO 40 I=1,M
      F0=AUX(I)*H
      I2=I+L
      F1=AUX(I2 )*H
      I3=I+L2
      F2=AUX(I3   )*H
      I4=I+L3
      F3=AUX(I4   )*H
      E(I)=(42.*F0+224.*F2+21.*F3)/336.
      AUX(I)=Y(I)+(7.*F0+10.*F1+F3)/27.
C*    AUX(I+L)=Y(I)+(28.*F0-125.*F1+546.*F2+54.*F3)/625.
          I00 = I + L
      AUX(I00)=Y(I)+(28.*F0-125.*F1+546.*F2+54.*F3)/625.
40    Y(I)=Y(I)+(F0+4.*F2+F3)/6.
      IF(L.EQ.M)GO TO 47
      M1=M+1
      H6=H/6.
      DO 45 I=M1,L
      I3=I+L2
      I4=I+L3
45    Y(I)=Y(I)+(AUX(I)+4.*AUX(I3   )+AUX(I4   ))*H6
47    CALL FUNC(T+2.*H/3.,AUX,AUX(L2+1),M)
      F4=-378.*H/625.
      DO 50 I=1,M
      I2=I+L
      I3=I+L2
50    AUX(I2 )=AUX(I2 )+AUX(I3   )*F4
      CALL FUNC(T+H/5.,AUX(L+1),AUX(L3+1),M)
      F4=162.*H/336.
      F5=125.*H/336.
      DO 60 I=1,M
      I3=I+L2
      I4=I+L3
60    E(I)=F4*AUX(I3   )+F5*AUX(I4   )-E(I)
      T=T+H
      RETURN
      END


      SUBROUTINE DD03DD(A,B,NI,MO,PP,PO,QI,DI,EPSI,TU,Y,YU,MV,KN,AUX)
C THIS SUBROUTINE (FORMERLY CALLED ONEPAS) OBTAINS FIRST APPROXIMATION
C TO Y AND FINDS THE SHOOTING AND MATCHING POINTS IF THESE ARE NOT GIVEN
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER NI,MO,PP,PO,QI,DI
      REAL*8 KN(NI,PP,QI),AUX(NI,PO,7),TU(2,QI),YU(NI,QI),MV(NI,MO,QI)
C INITIALIZE ERROR PROPAGATION MATRICES MV AND KN.
      LI=QI-1
      DO 101 KK=1,LI
      DO 101 JJ=1,NI
      DO 102 II=1,MO
 102  MV(JJ,II,KK)=0.0
      MV(JJ,JJ,KK)=1.0
      II=JJ+NI
      MV(JJ,II,KK)=-1.0
      DO 101 II=1,PP
 101  KN(JJ,II,KK)=0.0
      IF (DI.LT.1) GO TO 103
C OBTAIN FIRST APPROXIMATION TO Y IN CASE WHEN SHOOTING POINTS ARE GIVEN
      DO 104 KK=1,DI
104   CALL Y(TU(1,KK),YU(1,KK))
      DI=-DI
      RETURN
C CHOOSE SHOOTING AND MATCHING POINTS.
C
C SET UP FIRST SHOOTING POINT AND THE WHOLE INTERVAL (A,B).
 103  JK=1
      KK=QI
      TU(1,1)=A
      TU(2,QI)=B
      DT=B-A
      TU(1,QI)=DT
C OBTAIN ESTIMATE AT SHOOTING POINT.
 123  T=TU(1,JK)
      TT=TU(2,KK)
      CALL Y(T,YU(1,JK))
C INITIALIZE PARTIAL DERIVATIVE ARRAY AUX.
 112  DO 115 JJ=1,NI
      DO 116 II=2,PO
 116  AUX(JJ,II,1)=0.0
      AUX(JJ,JJ+1,1)=1.0
 115  AUX(JJ,1,1)=YU(JJ,JK)
      CALL DD03JD(.FALSE.,NI*PO,NI,T,DT,TT,AUX,EPSI,AUX(1,1,2))
C ADD RESULTS TO PROPAGATION MATRICES, INSERT MATCHING POINT AND
C REDUCE REMAINING INTERVAL.
      DO 117 JJ=1,NI
      KN(JJ,PP,JK)=KN(JJ,PP,JK)+AUX(JJ,1,1)
      IF (PP.LT.2) GO TO 118
      DO 119 II=2,PP
      K=NI+II
 119  KN(JJ,II-1,JK)=KN(JJ,II-1,JK)+AUX(JJ,K,1)
 118  DO 117 II=1,NI
 117  MV(JJ,II,JK)=AUX(JJ,II+1,1)
      TU(2,JK)=T
      TT=TT-T
      JK=JK+1
C JUMP IF REMAINING INTERVAL IS SMALL.
      IF (T.EQ.T+TT) GO TO 120
C JUMP IF ALL SHOOTING INTERVALS HAVE BEEN USED.
      IF (JK.GT.KK) GO TO 121
C JUMP IF THE FORWARD SHOOTING INTERVAL IS NOT GREATER THAN THE BACKWARD
C SHOOTING INTERVAL.
      IF ((T-TU(1,JK-1))/TU(1,KK).LE.1.0) GO TO 122
C INSERT NEW SHOOTING POINT.
      TU(1,JK)=T
      GO TO 123
C REVERSE THE INTEGRATION STEP.
 122  DT=-DT
C SET UP SHOOTING POINT FOR BACKWARD SHOOTING AND REMAINING BACKWARD
C INTERVAL.
 134  T=TU(2,KK)
      TU(1,KK)=T
      TT=TU(2,JK-1)
C OBTAIN ESTIMATE AT SHOOTING POINT.
      CALL Y(T,YU(1,KK))
C INITIALIZE PARTIAL DERIVATIVE ARRAY.
 126  DO 129 JJ=1,NI
      DO 130 II=2,PO
 130  AUX(JJ,II,1)=0.0
      AUX(JJ,JJ+1,1)=-1.0
 129  AUX(JJ,1,1)=YU(JJ,KK)
      KK=KK-1
      CALL DD03JD(.FALSE.,NI*PO,NI,T,DT,TT,AUX,EPSI,AUX(1,1,2))
C SUBTRACT RESULTS FROM PROPAGATION MATRICES, INSERT MATCHING POINT AND
C REDUCE REMAINING INTERVAL.
      DO 131 JJ=1,NI
      KN(JJ,PP,KK)=KN(JJ,PP,KK)-AUX(JJ,1,1)
      IF (PP.LT.2) GO TO 132
      DO 133 II=2,PP
      K=NI+II
 133  KN(JJ,II-1,KK)=KN(JJ,II-1,KK)-AUX(JJ,K,1)
 132  K=NI
      DO 131 II=1,NI
      K=K+1
 131  MV(JJ,K,KK)=AUX(JJ,II+1,1)
      TU(2,KK)=T
      TT=TT-T
C JUMP IF REMAINING INTERVAL IS SMALL.
      IF (T.EQ.T+TT) GO TO 120
C JUMP IF ALL SHOOTING POINTS HAVE BEEN USED.
      IF (JK.GT.KK) GO TO 121
      TT=TU(1,KK+1)-T
C JUMP IF THE BACKWARD INTERVAL WAS GREATER THAN THE FORWARD SHOOTING
C INTERVAL.
      IF ((TU(2,JK-1)-TU(1,JK-1))/TT.LT.1.0) GO TO 134
C SET UP SHOOTING POINT FOR FORWARD SHOOTING AND REMAINING FORWARD
C INTERVAL AND REVERSE INTEGRATION STEP.
      TU(1,KK)=TT
      TU(1,JK)=TU(2,JK-1)
      DT=-DT
      GO TO 123
C SET MARKER FOR INSUFFICIENT SHOOTING INTERVALS.
```

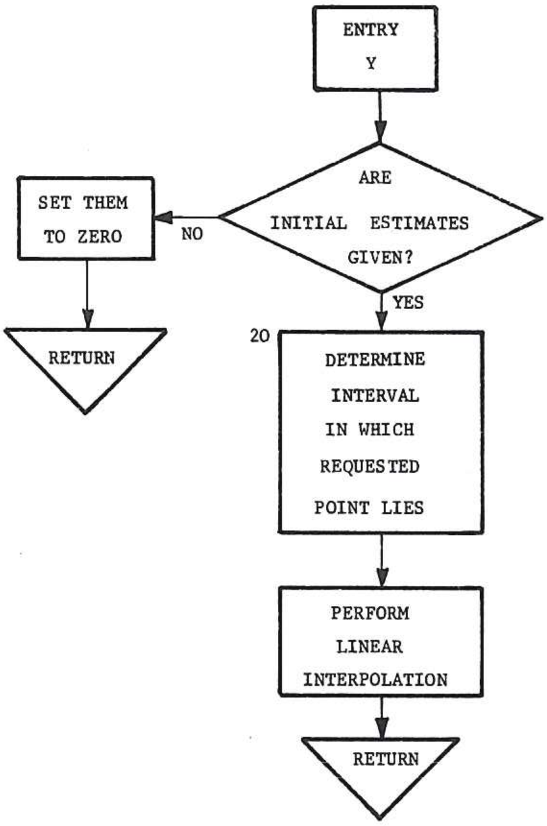
```
 121  DI=-1
C JUMP IF BACKWARD SHOOT HAS BEEN PERFORMED.
 120  IF (KK.LT.QI) GO TO 135
C OBTAIN ESTIMATE AT B.
      CALL Y(B,YU(1,KK))
C INSERT A NULL BACKWARD SHOOTING INTERVAL AND ADJUST PROPAGATION MATRIX
      TU(1,KK)=TU(2,KK)
      TU(2,KK-1)=TU(2,KK)
      KK=KK-1
      DO 141 JJ=1,NI
 141  KN(JJ,PP,KK)=KN(JJ,PP,KK)-YU(JJ,KK+1)
C JUMP IF ALL SHOOTING INTERVALS HAVE NOT BEEN USED.
 135  IF (JK.LE.KK) GO TO 142
C SET JK EQUAL TO NUMBER OF SHOOTING POINTS.
      JK=QI
      GO TO 143
C ADJUST PROPAGATION MATRIX AT FINAL MATCHING POINT.
 142  JK=JK-1
      DO 144 JJ=1,PP
      DO 144 II=1,NI
 144  KN(II,JJ,KK)=KN(II,JJ,KK)+KN(II,JJ,JK)
C ARRANGE IN COMPACT FORM THE PROPAGATION MATRICES AND ARRAYS DEFINING
C SHOOTING INTERVALS AND ESTIMATES.
      DO 145 JJ=KK,LI
      DO 146 II=1,NI
      DO 147 K=1,PP
 147  KN(II,K,JK)=KN(II,K,JJ)
      J=NI+1
      DO 146 K=J,MO
 146  MV(II,K,JK)=MV(II,K,JJ)
      JK=JK+1
      TU(1,JK)=TU(1,JJ+1)
      TU(2,JK)=TU(2,JJ+1)
      DO 145 II=1,NI
 145  YU(II,JK)=YU(II,JJ+1)
C RESET MARKER FOR FINAL MATCHING POINT.
      KK=KK+JK-QI
C SET DI EQUAL TO NUMBER OF SHOOTING POINTS, OR ZERO IF NOT ENOUGH.
 143  IF (DI.EQ.0) DI=JK
      IF (DI.LT.0) DI=0
C MAKE FINAL ADJUSTMENTS TO THE PROPAGATION MATRICES TO ACCOUNT FOR THE
C ALTERNATE NULL INTERVALS.
      JK=JK-1
      K=KK+1
      DO 148 JJ=1,NI
      IF (K.GT.JK) GO TO 149
      DO 150 II=K,JK
 150  KN(JJ,PP,II)=KN(JJ,PP,II)+YU(JJ,II)
 149  IF (KK.LT.2) GO TO 148
      DO 151 II=2,KK
 151  KN(JJ,PP,II-1)=KN(JJ,PP,II-1)-YU(JJ,II)
 148  CONTINUE
 109  RETURN
      END


      SUBROUTINE DD03ED(NI,YMUST,G,MU,AUX)
C THIS IS AN INITIALIZATION ENTRY.
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER NI,PO
      REAL*8 MU(1),AUX(1)
      GO TO 1
C THIS ENTRY IS CALLED BY THE RUNGE-KUTTA SUBROUTINE DD03I
      ENTRY DD03FD(T,Y,VF,LO)
      DIMENSION VF(1),Y(1)
      CALL G(T,Y,MU,VF,VF(NI+1),NI)
C RETURN IF VARIATIONAL DERIVATIVES NOT WANTED.
      IF(NI.EQ.LO)RETURN
      PO=LO/NI
C JUMP IF ANALYTIC DERIVATIVES ARE AVAILABLE.
```

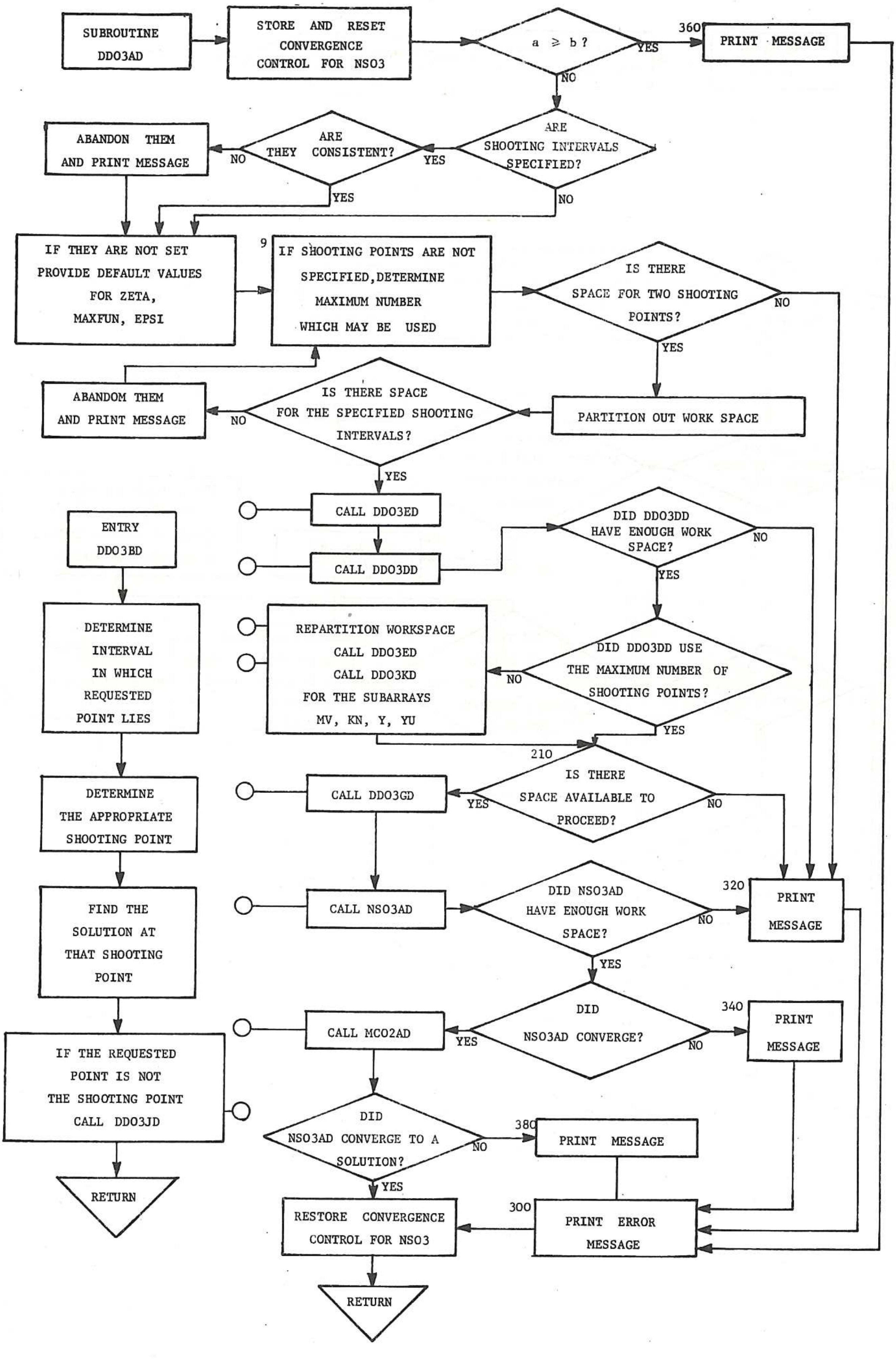
```
      IF (YMUST.EQ.0.0) GO TO 2
C FOR EACH VARIABLE,ADD FINITE INCREMENT AND FIND ONE-SIDED DIFFERENCE
      K=NI+1
      DO 3 I=2,PO
      DO 4 J=1,NI
      AUX(J)=Y(J)+Y(K)*YMUST
 4    K=K+1
      K=K-NI
      J=I-1-NI
      IF (J.LE.0) GO TO 5
      S=MU(J)
      MU(J)=S+YMUST
 5    CALL G(T,AUX,MU,VF(K),DUMMY,NI)
      IF (J.GT.0) MU(J)=S
      DO 3 J=1,NI
      VF(K)=(VF(K)-VF(J))/YMUST
 3    K=K+1
      GO TO 1
C CALCULATE THE DIFFERENTIALS FROM THE PARTIAL DERIVATIVES SUPLLIED BY G
2     DO  9 I=1,NI
      L=I
      DO 7 K=1,NI
      L=L+NI
7     AUX(K)=VF(L)
      L=I
      DO 9 J=2,PO
      SUM=0.
      L=L+NI
      IF(J.GT.NI+1)SUM=VF(L)
      M=NI*J-NI
      DO 8 K=1,NI
      M=M+1
8     SUM=SUM+AUX(K)*Y(M)
9     VF(L)=SUM
 1    RETURN
      END
```

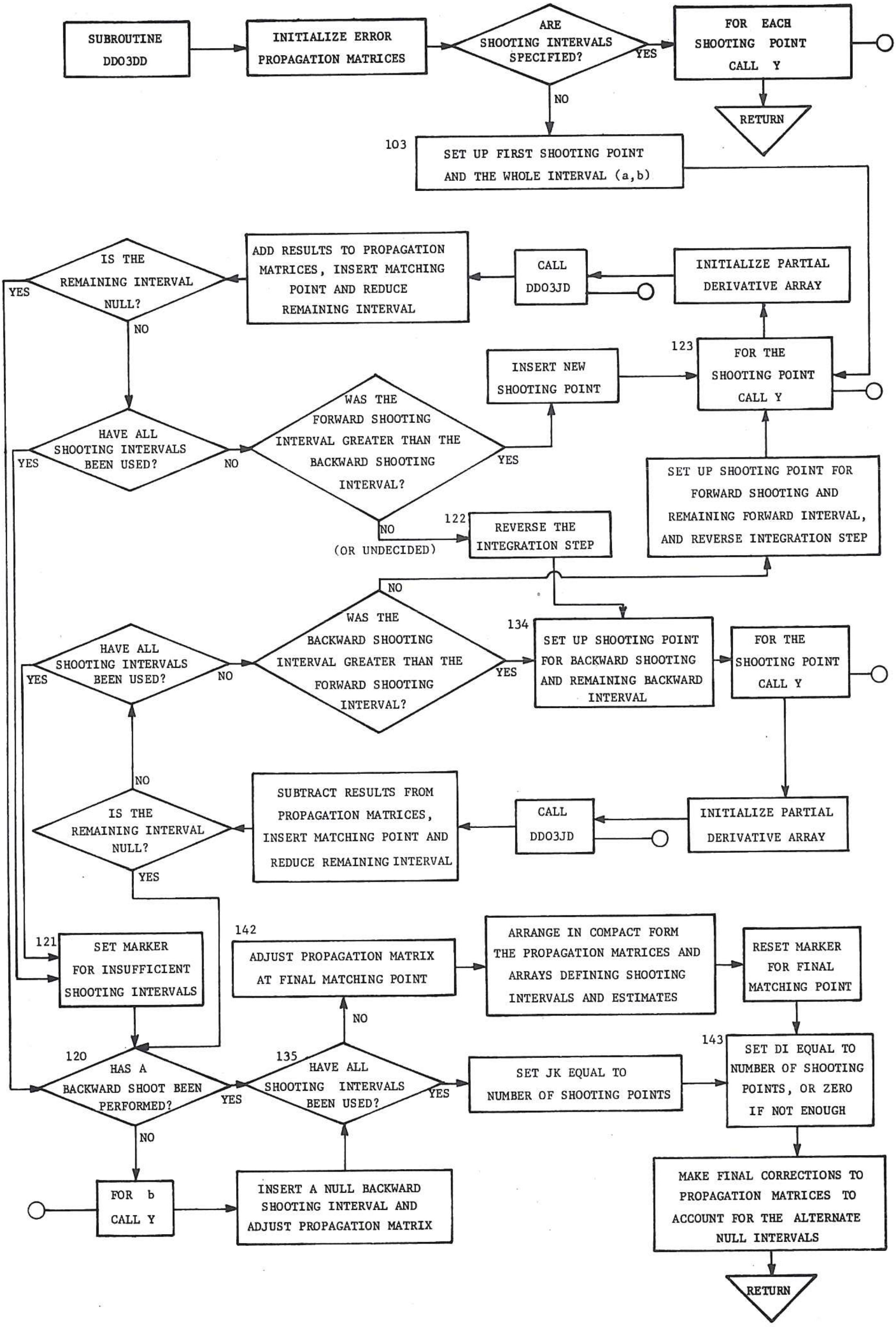
APPENDIX B. Flow Charts

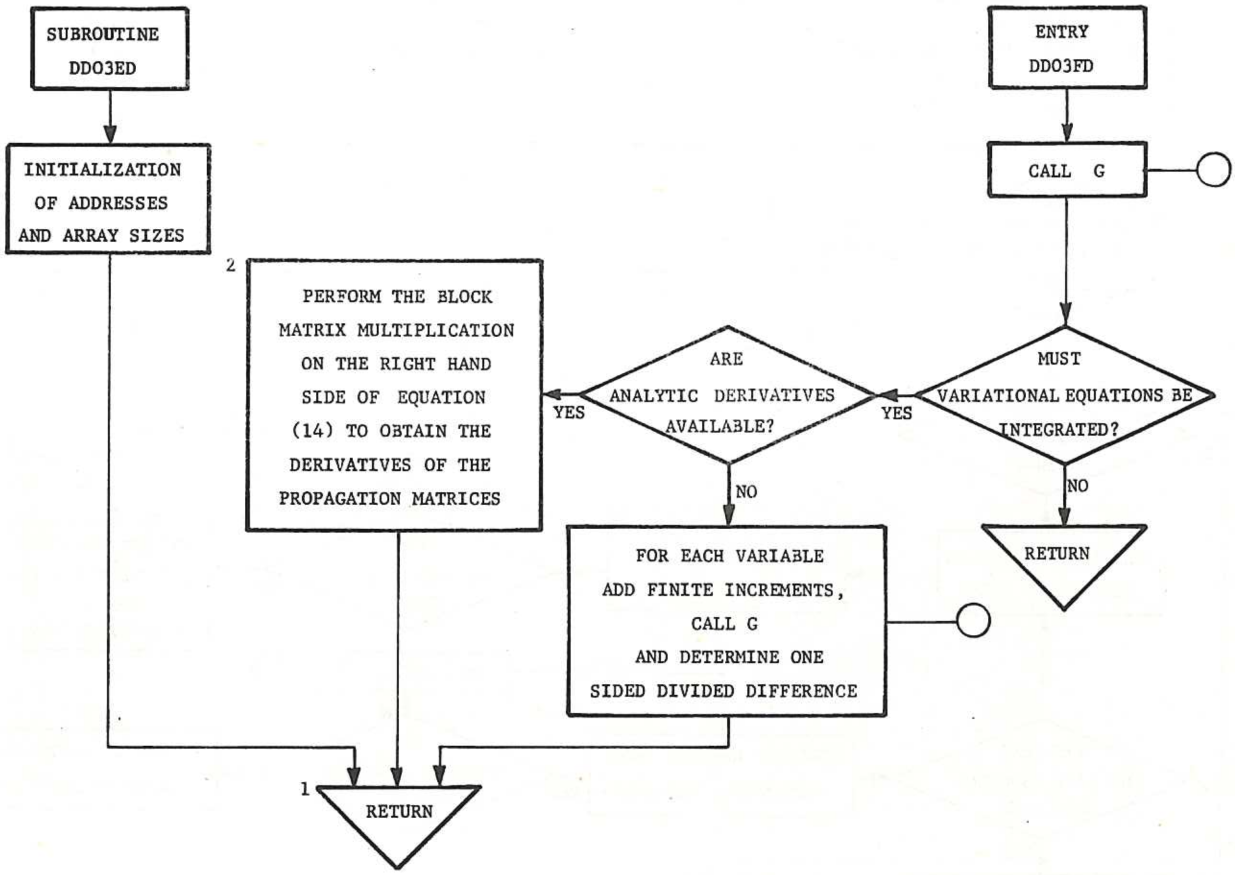

SUBROUTINE SETDEF
INITIALIZATION OF ADDRESSES AND DEFAULT VALUES
INITIALIZATION OF INTEGERS NEEDED BY DEFAULT ROUTINE H
READING OF SCALAR DATA (&VALUES)
RETURN
ENTRY PROGC
INITIALIZATION OF μ IF NOT SET IN DATA
PRINT DETAILS OF INPUT DATA
CALL SETY
CALL DD03AD
WAS ENOUGH WORK SPACE PROVIDED?
NO
YES
ARE TABULATION POINTS SPECIFIED?
YES
SET TABULATION POINTS AS SPECIFIED
NO
60
SET TABULATION POINTS FROM SHOOTING POINTS
80
FOR EACH TABULATION POINT CALL DD03BD
100
PRINT BEST RESULTS OBTAINED
RETURN

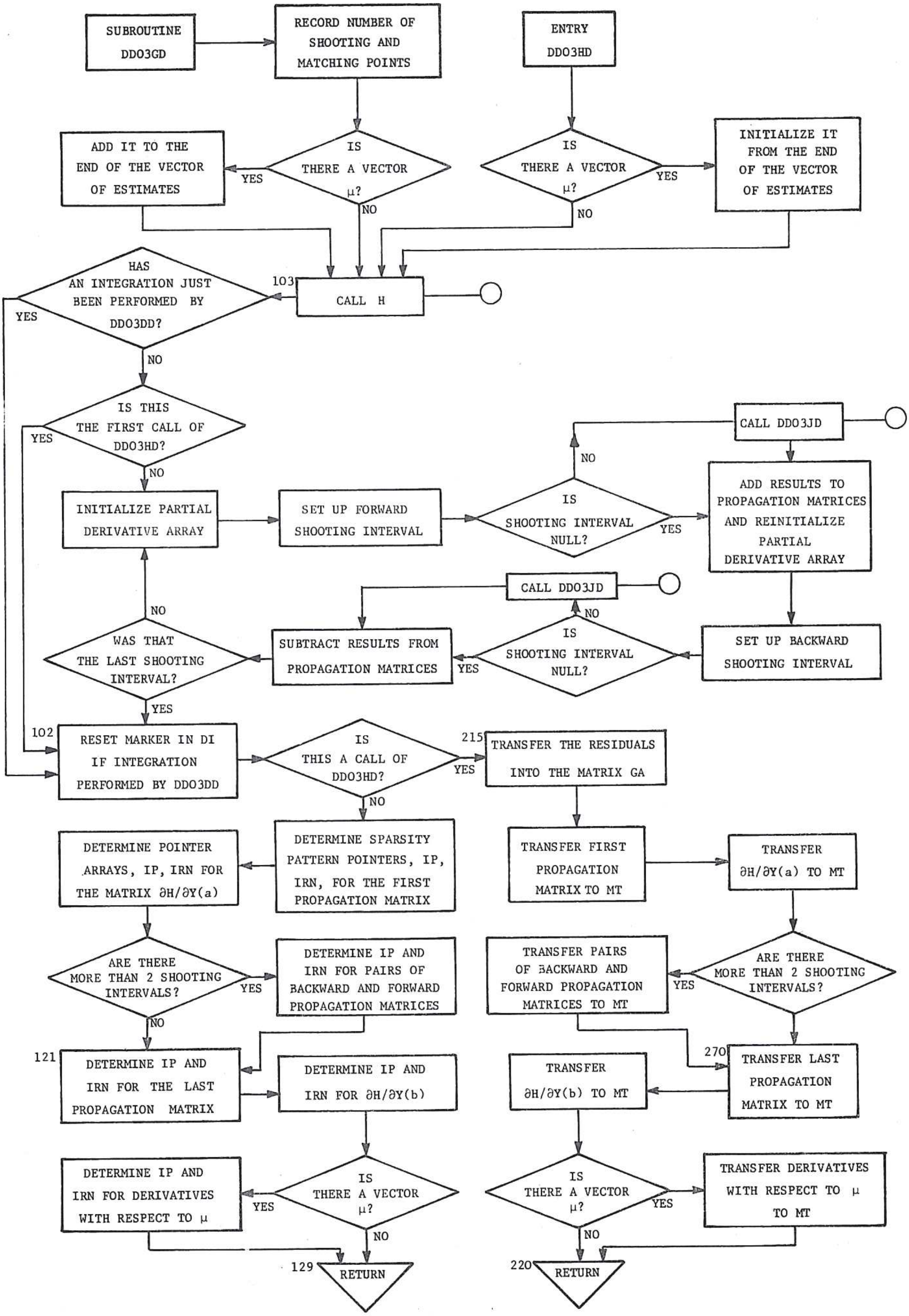

SUBROUTINE SETY
INITIALIZATION OF ADDRESSES AND DIMENSIONS
RETURN

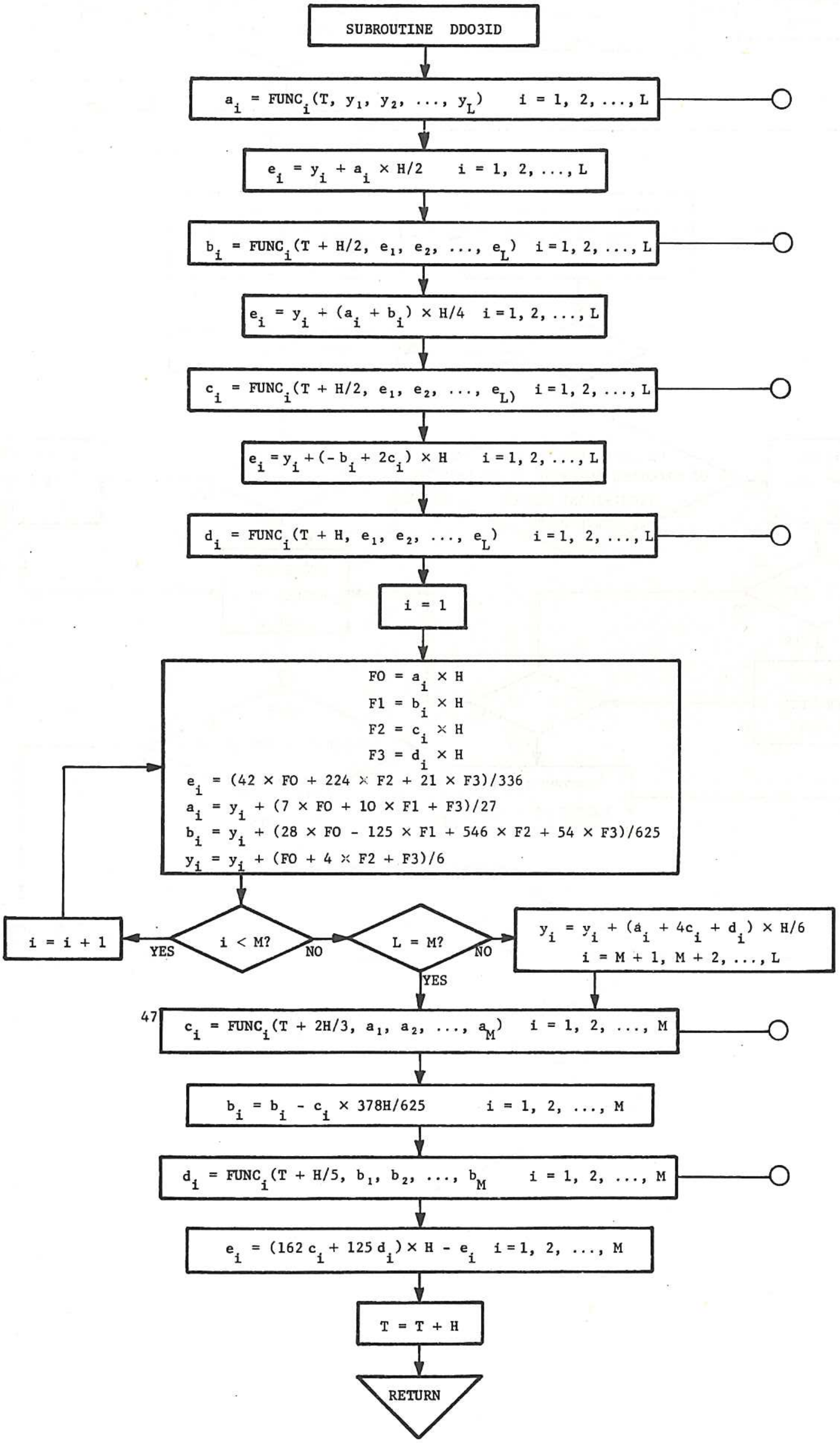

ENTRY Y
ARE INITIAL ESTIMATES GIVEN?
NO
SET THEM TO ZERO
RETURN
YES
20
DETERMINE INTERVAL IN WHICH REQUESTED POINT LIES
PERFORM LINEAR INTERPOLATION
RETURN

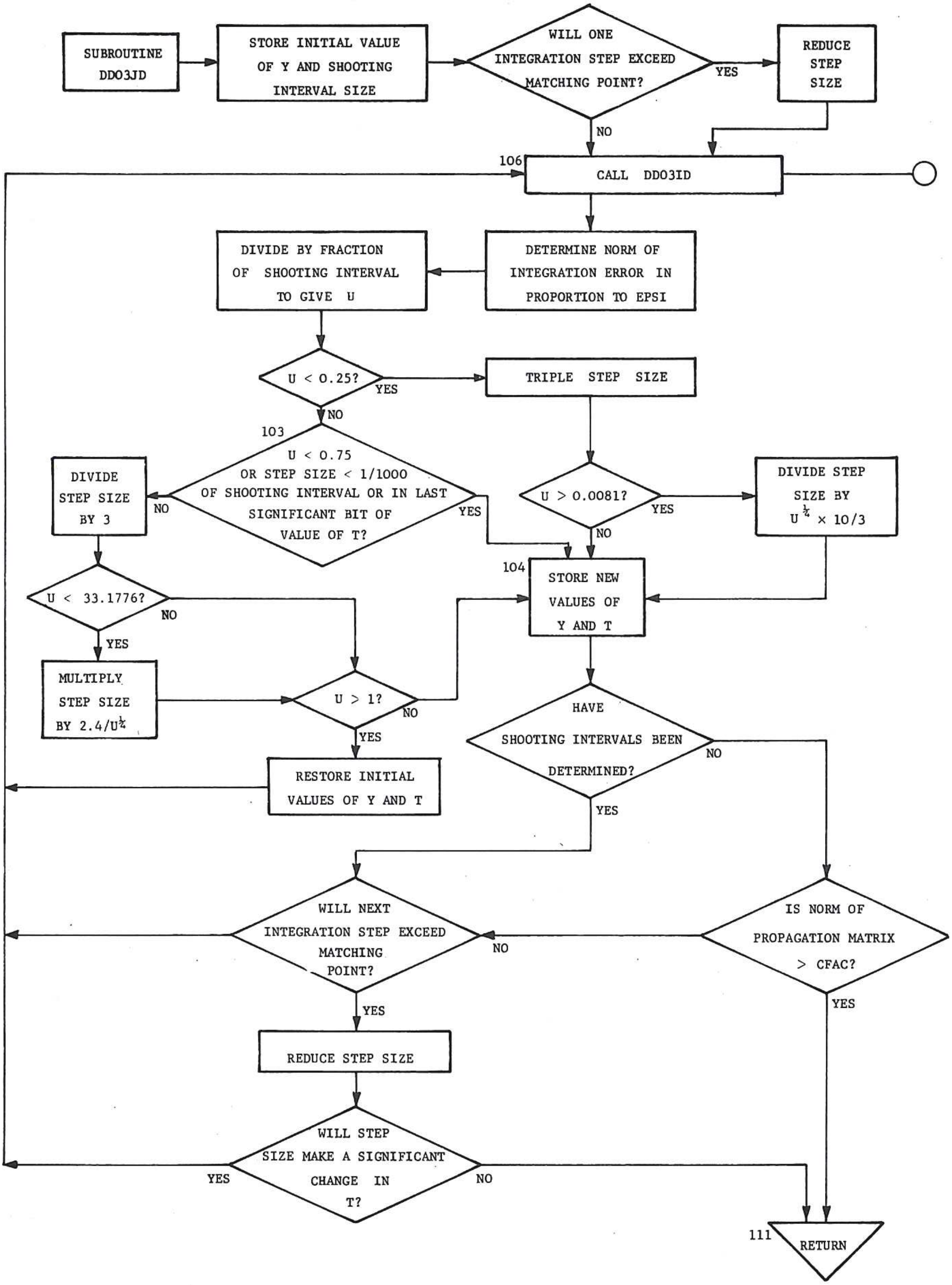

SUBROUTINE DD03AD
STORE AND RESET CONVERGENCE CONTROL FOR NS03
a ≥ b ?
YES
NO
360
PRINT MESSAGE
ARE SHOOTING INTERVALS SPECIFIED?
YES
NO
ARE THEY CONSISTENT?
NO
YES
ABANDON THEM AND PRINT MESSAGE
IF THEY ARE NOT SET PROVIDE DEFAULT VALUES FOR ZETA, MAXFUN, EPSI
9
IF SHOOTING POINTS ARE NOT SPECIFIED, DETERMINE MAXIMUM NUMBER WHICH MAY BE USED
IS THERE SPACE FOR TWO SHOOTING POINTS?
NO
YES
PARTITION OUT WORK SPACE
IS THERE SPACE FOR THE SPECIFIED SHOOTING INTERVALS?
NO
YES
ABANDON THEM AND PRINT MESSAGE
CALL DD03ED
CALL DD03DD
DID DD03DD HAVE ENOUGH WORK SPACE?
NO
YES
DID DD03DD USE THE MAXIMUM NUMBER OF SHOOTING POINTS?
NO
YES
REPARTITION WORKSPACE CALL DD03ED CALL DD03KD FOR THE SUBARRAYS MV, KN, Y, YU
210
IS THERE SPACE AVAILABLE TO PROCEED?
YES
NO
CALL DD03GD
CALL NS03AD
DID NS03AD HAVE ENOUGH WORK SPACE?
NO
YES
320
PRINT MESSAGE
DID NS03AD CONVERGE?
YES
NO
340
PRINT MESSAGE
CALL MC02AD
DID NS03AD CONVERGE TO A SOLUTION?
NO
YES
380
PRINT MESSAGE
300
PRINT ERROR MESSAGE
RESTORE CONVERGENCE CONTROL FOR NS03
RETURN
ENTRY DD03BD
DETERMINE INTERVAL IN WHICH REQUESTED POINT LIES
DETERMINE THE APPROPRIATE SHOOTING POINT
FIND THE SOLUTION AT THAT SHOOTING POINT
IF THE REQUESTED POINT IS NOT THE SHOOTING POINT CALL DD03JD
RETURN


SUBROUTINE DD03DD
INITIALIZE ERROR PROPAGATION MATRICES
ARE SHOOTING INTERVALS SPECIFIED?
YES
FOR EACH SHOOTING POINT CALL Y
RETURN
NO
103
SET UP FIRST SHOOTING POINT AND THE WHOLE INTERVAL (a,b)
IS THE REMAINING INTERVAL NULL?
YES
NO
ADD RESULTS TO PROPAGATION MATRICES, INSERT MATCHING POINT AND REDUCE REMAINING INTERVAL
CALL DD03JD
INITIALIZE PARTIAL DERIVATIVE ARRAY
123
FOR THE SHOOTING POINT CALL Y
INSERT NEW SHOOTING POINT
HAVE ALL SHOOTING INTERVALS BEEN USED?
YES
NO
WAS THE FORWARD SHOOTING INTERVAL GREATER THAN THE BACKWARD SHOOTING INTERVAL?
YES
NO
(OR UNDECIDED)
122
REVERSE THE INTEGRATION STEP
SET UP SHOOTING POINT FOR FORWARD SHOOTING AND REMAINING FORWARD INTERVAL, AND REVERSE INTEGRATION STEP
NO
HAVE ALL SHOOTING INTERVALS BEEN USED?
YES
NO
WAS THE BACKWARD SHOOTING INTERVAL GREATER THAN THE FORWARD SHOOTING INTERVAL?
YES
134
SET UP SHOOTING POINT FOR BACKWARD SHOOTING AND REMAINING BACKWARD INTERVAL
FOR THE SHOOTING POINT CALL Y
NO
IS THE REMAINING INTERVAL NULL?
YES
SUBTRACT RESULTS FROM PROPAGATION MATRICES, INSERT MATCHING POINT AND REDUCE REMAINING INTERVAL
CALL DD03JD
INITIALIZE PARTIAL DERIVATIVE ARRAY
121
SET MARKER FOR INSUFFICIENT SHOOTING INTERVALS
142
ADJUST PROPAGATION MATRIX AT FINAL MATCHING POINT
ARRANGE IN COMPACT FORM THE PROPAGATION MATRICES AND ARRAYS DEFINING SHOOTING INTERVALS AND ESTIMATES
RESET MARKER FOR FINAL MATCHING POINT
120
HAS A BACKWARD SHOOT BEEN PERFORMED?
YES
NO
135
HAVE ALL SHOOTING INTERVALS BEEN USED?
NO
YES
SET JK EQUAL TO NUMBER OF SHOOTING POINTS
143
SET DI EQUAL TO NUMBER OF SHOOTING POINTS, OR ZERO IF NOT ENOUGH
FOR b CALL Y
INSERT A NULL BACKWARD SHOOTING INTERVAL AND ADJUST PROPAGATION MATRIX
MAKE FINAL CORRECTIONS TO PROPAGATION MATRICES TO ACCOUNT FOR THE ALTERNATE NULL INTERVALS
RETURN


SUBROUTINE DD03ED
INITIALIZATION OF ADDRESSES AND ARRAY SIZES
2
PERFORM THE BLOCK MATRIX MULTIPLICATION ON THE RIGHT HAND SIDE OF EQUATION (14) TO OBTAIN THE DERIVATIVES OF THE PROPAGATION MATRICES
ENTRY DD03FD
CALL G
MUST VARIATIONAL EQUATIONS BE INTEGRATED?
YES
NO
RETURN
ARE ANALYTIC DERIVATIVES AVAILABLE?
YES
NO
FOR EACH VARIABLE ADD FINITE INCREMENTS, CALL G AND DETERMINE ONE SIDED DIVIDED DIFFERENCE
1
RETURN


SUBROUTINE DD03GD
RECORD NUMBER OF SHOOTING AND MATCHING POINTS
ENTRY DD03HD
ADD IT TO THE END OF THE VECTOR OF ESTIMATES
YES
IS THERE A VECTOR μ?
NO
IS THERE A VECTOR μ?
YES
NO
INITIALIZE IT FROM THE END OF THE VECTOR OF ESTIMATES
HAS AN INTEGRATION JUST BEEN PERFORMED BY DD03DD?
YES
NO
103
CALL H
IS THIS THE FIRST CALL OF DD03HD?
YES
NO
CALL DD03JD
NO
INITIALIZE PARTIAL DERIVATIVE ARRAY
SET UP FORWARD SHOOTING INTERVAL
IS SHOOTING INTERVAL NULL?
YES
ADD RESULTS TO PROPAGATION MATRICES AND REINITIALIZE PARTIAL DERIVATIVE ARRAY
CALL DD03JD
NO
NO
WAS THAT THE LAST SHOOTING INTERVAL?
SUBTRACT RESULTS FROM PROPAGATION MATRICES
YES
IS SHOOTING INTERVAL NULL?
SET UP BACKWARD SHOOTING INTERVAL
YES
102
RESET MARKER IN DI IF INTEGRATION PERFORMED BY DD03DD
IS THIS A CALL OF DD03HD?
YES
215
TRANSFER THE RESIDUALS INTO THE MATRIX GA
NO
DETERMINE POINTER ARRAYS, IP, IRN FOR THE MATRIX ∂H/∂Y(a)
DETERMINE SPARSITY PATTERN POINTERS, IP, IRN, FOR THE FIRST PROPAGATION MATRIX
TRANSFER FIRST PROPAGATION MATRIX TO MT
TRANSFER ∂H/∂Y(a) TO MT
ARE THERE MORE THAN 2 SHOOTING INTERVALS?
YES
DETERMINE IP AND IRN FOR PAIRS OF BACKWARD AND FORWARD PROPAGATION MATRICES
TRANSFER PAIRS OF BACKWARD AND FORWARD PROPAGATION MATRICES TO MT
YES
ARE THERE MORE THAN 2 SHOOTING INTERVALS?
NO
121
DETERMINE IP AND IRN FOR THE LAST PROPAGATION MATRIX
DETERMINE IP AND IRN FOR ∂H/∂Y(b)
TRANSFER ∂H/∂Y(b) TO MT
270
TRANSFER LAST PROPAGATION MATRIX TO MT
DETERMINE IP AND IRN FOR DERIVATIVES WITH RESPECT TO μ
YES
IS THERE A VECTOR μ?
NO
IS THERE A VECTOR μ?
YES
TRANSFER DERIVATIVES WITH RESPECT TO μ TO MT
NO
129
RETURN
220
RETURN


SUBROUTINE DD03ID
a_i = FUNC$_i$(T, y_1, y_2, ..., y_L)   i = 1, 2, ..., L
e_i = y_i + a_i × H/2   i = 1, 2, ..., L
b_i = FUNC$_i$(T + H/2, e_1, e_2, ..., e_L)   i = 1, 2, ..., L
e_i = y_i + (a_i + b_i) × H/4   i = 1, 2, ..., L
c_i = FUNC$_i$(T + H/2, e_1, e_2, ..., $e_{L)}$ i = 1, 2, ..., L
e_i = y_i + (- b_i + 2c_i) × H   i = 1, 2, ..., L
d_i = FUNC$_i$(T + H, e_1, e_2, ..., e_L)   i = 1, 2, ..., L
i = 1
F0 = a_i × H
F1 = b_i × H
F2 = c_i × H
F3 = d_i × H
e_i = (42 × F0 + 224 × F2 + 21 × F3)/336
a_i = y_i + (7 × F0 + 10 × F1 + F3)/27
b_i = y_i + (28 × F0 - 125 × F1 + 546 × F2 + 54 × F3)/625
y_i = y_i + (F0 + 4 × F2 + F3)/6
i = i + 1
YES
i < M?
NO
L = M?
NO
YES
y_i = y_i + (a_i + 4c_i + d_i) × H/6
i = M + 1, M + 2, ..., L
47
c_i = FUNC$_i$(T + 2H/3, a_1, a_2, ..., a_M)   i = 1, 2, ..., M
b_i = b_i - c_i × 378H/625   i = 1, 2, ..., M
d_i = FUNC$_i$(T + H/5, b_1, b_2, ..., b_M i = 1, 2, ..., M
e_i = (162 c_i + 125 d_i) × H - e_i i = 1, 2, ..., M
T = T + H
RETURN


SUBROUTINE DD03JD
STORE INITIAL VALUE OF Y AND SHOOTING INTERVAL SIZE
WILL ONE INTEGRATION STEP EXCEED MATCHING POINT?
YES
REDUCE STEP SIZE
NO
106
CALL DD03ID
DETERMINE NORM OF INTEGRATION ERROR IN PROPORTION TO EPSI
DIVIDE BY FRACTION OF SHOOTING INTERVAL TO GIVE U
U < 0.25?
YES
TRIPLE STEP SIZE
NO
103
U < 0.75 OR STEP SIZE < 1/1000 OF SHOOTING INTERVAL OR IN LAST SIGNIFICANT BIT OF VALUE OF T?
NO
YES
DIVIDE STEP SIZE BY 3
U > 0.0081?
YES
DIVIDE STEP SIZE BY U¼ × 10/3
NO
104
STORE NEW VALUES OF Y AND T
U < 33.1776?
NO
YES
MULTIPLY STEP SIZE BY 2.4/U¼
U > 1?
NO
YES
RESTORE INITIAL VALUES OF Y AND T
HAVE SHOOTING INTERVALS BEEN DETERMINED?
NO
YES
WILL NEXT INTEGRATION STEP EXCEED MATCHING POINT?
NO
IS NORM OF PROPAGATION MATRIX > CFAC?
YES
YES
REDUCE STEP SIZE
WILL STEP SIZE MAKE A SIGNIFICANT CHANGE IN T?
YES
NO
111
RETURN

APPENDIX C. OUTPUT FROM SAMPLE PROBLEM

```
                                                      NUMBER OF DEPENDENT VARIABLES Y: N=    5

                                                    NUMBER OF BOUNDARY CONDITIONS: N+P=NO=   6

                    NUMBER OF VARIABLES INVOLVED IN BOUNDARY CONDITIONS: 2N+P+1= NN=   12

                                             NUMBER OF WORDS OF WORK SPACE PROVIDED: ITU=  3464

                                                            DIMENSION OF EIGENVALUE MU: P=   1

                     MAXIMUM STORAGE FOR TABULATION POINTS OR ESTIMATION POINTS: R=  10

                  NUMBER OF ESTIMATED VALUES OF Y (POSITIVE IF MU IS ESTIMATED): RI=  -2

                                                 NUMBER OF SPECIFIED TABULATION POINTS: SI=   0

                          VALUES OF INDEPENDENT VARIABLE T AT BOUNDARY POINTS: A=    0.0000
                                                                               B=   18.0000

                          DEFAULT VALUES FOR JACOBIAN MATRIX DH
                  OR COEFFICIENTS OF BOUNDARY CONDITIONS H1,H2,H3,H0

    1    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    2    0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
    3    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    4    0.0000    0.0000    1.0000    0.0000    0.0000    0.0000
    5    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    6    0.0000    0.0000    0.0000    1.0000    0.0000    0.0000
    7    0.0000    0.0000    0.0000    0.0000    1.0000    0.0000
    8    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    9    0.0000    0.0000    0.0000    0.0000    0.0000    1.0000
   10    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   11    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   12    0.0000    0.0000    1.0000    0.0000    0.0000    0.5000

ESTIMATED VALUES OF Y(   0.000000)  0.0000D 00  0.0000D 00  0.0000D 00  1.0000D 00  0.0000D 00

ESTIMATED VALUES OF Y(  18.000000)  0.0000D 00  0.0000D 00  0.0000D 00  0.0000D 00  0.0000D 00

                                            NUMBER OF ITERATIONS BETWEEN PRINTOUTS: LPRINT=   1

                                            MAXIMUM NUMBER OF INTEGRATIONS ALLOWED: MAXFUN=  40

                  INCREMENT IN Y AND MU FOR DIFFERENTIATION OF THE FUNCTION G: YMUST=  0.00D 00

                                                      ERROR BOUND ON SOLUTION: ZETA=  1.00D-06
                    ERROR BOUND FOR THE DEPENDENT VARIABLES Y DURING INTEGRATION: EPSI= -1.00D-04

10 SPECIFIED SHOOTING POINTS

      0.0000     2.0000
      2.0000     4.0000
      4.0000     6.0000
      6.0000     8.0000
      8.0000    10.0000
     10.0000    12.0000
     12.0000    14.0000
     14.0000    16.0000
     16.0000    18.0000
     18.0000
AFTER   1 ITERATIONS AND   1 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  5.3214D 01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  1.8758D 00  0.0000D 00
THE NORM OF THE VECTOR V IS  1.4854D 02
AFTER   2 ITERATIONS AND   3 CALLS OF FUNC LAMDA IS  2.1970D-03 AND THE SUM OF SQUARES OF RESIDUALS IS  5.3214D 01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  1.8758D 00  8.1824D 00
THE NORM OF THE VECTOR V IS  1.4854D 02
AFTER   3 ITERATIONS AND   5 CALLS OF FUNC LAMDA IS  1.1981D-02 AND THE SUM OF SQUARES OF RESIDUALS IS  5.3214D 01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  1.8758D 00  3.6874D 00
THE NORM OF THE VECTOR V IS  1.4854D 02
AFTER   4 ITERATIONS AND   6 CALLS OF FUNC LAMDA IS  1.0981D-01 AND THE SUM OF SQUARES OF RESIDUALS IS  4.9563D 01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  3.4060D 00  2.7709D 00
THE NORM OF THE VECTOR V IS  9.1121D 01
AFTER   5 ITERATIONS AND   7 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  4.9359D 00
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  3.0860D 00  1.8180D 00
THE NORM OF THE VECTOR V IS  1.5830D 01
AFTER   6 ITERATIONS AND   8 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  6.8036D-01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.5943D 00  1.8576D 00
THE NORM OF THE VECTOR V IS  4.2570D 00
AFTER   7 ITERATIONS AND   9 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  1.0559D-01
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.3746D 00  9.1427D-01
THE NORM OF THE VECTOR V IS  2.2632D 00
AFTER   8 ITERATIONS AND  10 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  5.5604D-04
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.5507D 00  2.8949D-01
THE NORM OF THE VECTOR V IS  1.7826D-01
AFTER   9 ITERATIONS AND  11 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  2.7990D-07
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.5657D 00  3.5214D-02
THE NORM OF THE VECTOR V IS  4.8389D-03
AFTER  10 ITERATIONS AND  12 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  3.9944D-14
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.5653D 00  5.8920D-04
THE NORM OF THE VECTOR V IS  2.0159D-06
AFTER  11 ITERATIONS AND  13 CALLS OF FUNC LAMDA IS  0.0000D 00 AND THE SUM OF SQUARES OF RESIDUALS IS  6.4228D-24
THE NORMS OF THE CURRENT ITERATE X AND THE LAST CHANGE MADE TO IT ARE  2.5653D 00  2.4481D-07
THE NORM OF THE VECTOR V IS  1.9234D-11
```

```
TABULATED VALUES OF Y(   0.000000)  2.4462D-19 -2.7338D-19  2.4372D-01  1.0000D 00 -2.5162D-01
TABULATED VALUES OF Y(   2.000000) -2.4791D-01  3.9895D-02 -4.3426D-02  7.2015D-01 -2.8845D-02
TABULATED VALUES OF Y(   4.000000) -2.9708D-01 -1.1552D-03 -3.2680D-03  7.1923D-01  6.2689D-03
TABULATED VALUES OF Y(   6.000000) -2.9120D-01 -6.5539D-04  8.8323D-04  7.2477D-01  3.8322D-04
TABULATED VALUES OF Y(   8.000000) -2.9041D-01 -9.4167D-05 -1.5353D-04  7.2469D-01 -1.6689D-04
TABULATED VALUES OF Y(  10.000000) -2.8929D-01 -3.4466D-04  2.5074D-04  7.2388D-01 -7.4209D-04
TABULATED VALUES OF Y(  12.000000) -2.9213D-01  2.7203D-03  2.9222D-03  7.2334D-01  1.4687D-03
TABULATED VALUES OF Y(  14.000000) -3.1096D-01  3.4884D-03 -7.4560D-03  7.3626D-01  1.1259D-02
TABULATED VALUES OF Y(  16.000000) -2.4433D-01 -4.9757D-02 -4.2123D-02  7.3225D-01 -3.7511D-02
TABULATED VALUES OF Y(  18.000000) -4.6881D-19 -5.0052D-19  1.8548D-01  5.0000D-01 -1.7881D-01
                                    TABULATED VALUE OF EIGENVALUE MU

          5.2491D-01

10 ACTUAL SHOOTING POINTS

          0.0000     2.0000
          2.0000     4.0000
          4.0000     6.0000
          6.0000     8.0000
          8.0000    10.0000
         10.0000    12.0000
         12.0000    14.0000
         14.0000    16.0000
         16.0000    18.0000
         18.0000

 2476 WORDS OF WORKSPACE USED
```