2

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-479

An Integrator Design

Fred T. Krogh

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

May 15, 1971

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-479

An Integrator Design

Fred T. Krogh

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

May 15, 1971

Prepared Under Contract No. NAS 7-100
National Aeronautics and Space Administration

PREFACE

The work described in this report was performed by the Data Systems Division of the Jet Propulsion Laboratory.

TABLE OF CONTENTS

ABSTRACT

The general design of a system of subroutines for solving the initial value problem in ordinary differential equations is given. An attempt has been made to design these subroutines in such a way that they will be easy to use on easy problems and still be flexible enough to treat any type of initial value problem with a high degree of efficiency. Emphasis is on the use of these subroutines, rather than on the mathematical algorithms which at this time are not completely specified. Implementation of our design in FORTRAN IV suffers from deficiencies in the design of the multiple entry feature provided in some of the current FORTRAN IV compilers.

I. INTRODUCTION

This paper gives the form in which we plan to provide subroutines for the numerical solution of ordinary differential equations. Our goal is to provide a reliable algorithm, which incorporates features of use in a wide variety of applications, which is highly efficient on a wide class of problems, and which is easy to use on easy problems. This document has two primary purposes.

1. We are soliciting comments from users and numerical analysts on the following: features not included which someone may find useful, features provided in a way more awkward than necessary, and the general approach.

2. We want to call attention to the sad state of the multiple entry feature provided in the FORTRAN compilers of third generation computers. None that we know of can be trusted to work as it should with the kind of usage outlined below. (The problem and some possible solutions will be sent on request.) Past experience with our UNIVAC 1108 indicates that with a little care (i.e. code which takes into account quirks in the compiler) we can get the desired results. On the IBM 360 things are a little worse, since in this case the end user will have to use tricks in his coding to get the desired results, and the higher the degree of optimization, the trickier he may have to be. The CDC 6000 series provides a multiple entry in name only, and thus the usage given below is not even permitted. Clearly it is poor algorithm design to be so computer dependent, and yet the other choices are so unsatisfactory that we take the approach below in the hope that the future will find the most widely used scientific programming language is one which has a well defined multiple entry feature. (Anyone for PL/1?)

In addition we hope that there may be some ideas here of use to others who are engaged in the design of mathematical software.

The design given here extends and modifies the features available in the integrator described in [1]. The main extensions are the special provisions for stiff equations and for variational equations.

The overall design has resulted from experience with the integrator in [1] and all of the features included here are of use in some application. The features are given in all of their gory detail, since we believe the detail necessary to the evaluation of how a feature will work in practice. The reader should, of course, skip over the details of features in which he has no interest. Results obtained with the integrator in [1] are given in [2].

Central to our approach is the use of reverse communication, which means that whenever additional information is to be communicated, a subroutine returns control to the program which called it. This is opposed to the common practice in integration subroutines of passing the name of the subprogram which computes the derivatives through the calling sequence of the integration subroutine, and then having the integration subroutine call the subprogram of this name whenever a derivative evaluation is required. Those who have experience only with the latter procedure may not be impressed with the facts that reverse communication may mean a little less fussing with CØMMØN, that it clarifies the structure of a large program, and that on simple problems only one program need be written. We adopted the reverse communication concept because Charles Lawson had made it the policy for subroutines in the JPL subroutine library. After trying both approaches we prefer to use reverse communication, and we believe that most people who give a fair trial to both approaches will feel the same way.

The following section lists the subroutines, gives their function, their entry points, the arguments passed through each entry point, and the reason for each entry point. This skeleton outline is meant only to serve as a guide to Sections III and IV which describe how to use these subroutines. Also see Figure 1 which outlines how the various subroutines communicate.

The remainder of this introduction gives the class of problems which the subroutines can be used to solve and gives a very brief sketch of the formulas which are used to effect the solution. The details here need not be completely understood in order to comprehend most of what follows.

The primary problem is the initial-value problem

$$z_i^{(d_i)} = f_i(t,\vec{y}) \quad \vec{y}(t_0) = \vec{\eta},\ i=1,2,\ldots,m \tag{1.1}$$

where $z_i^{(d_i)}$ is the d_i-th derivative of z_i with respect to t, and $\vec{y} = (z_1, z_1', \ldots, z_1^{(d_1-1)}, z_2, \ldots, z_m^{(d_m-1)})^T$. More generally we treat problems where everything in eq. (1.1) remains the same, except that $z_i^{(d_i)}$ is replaced by zero for one or more values of i. For such values of i, one has an algebraic equation (d_i=1) or an implicit differential equation ($d_i > 1$).

To simplify the description of the formulas, we consider only a single equation

$$z^{(d)} = f(t,\vec{y}) \quad \vec{y}(t_0) = \vec{\eta}, \quad \vec{y} = (z, z', \ldots, z^{(d-1)})^T \tag{1.2}$$

Of course, the formulas we give for eq. (1.2) apply to eq. (1.1) simply by applying these formulas to each equation in eq. (1.1). Let

$$\left.\begin{aligned}
&z_n = z(t_n) = z(t_0+nh)\\
&z_n^{(i)} = \left.\frac{d^i z}{dt^i}\right|_{t=t_n}\\
&p_n^{(i)} = \text{the predicted value of } z_n^{(i)}\\
&\nabla^k z_n^{(i)} = \begin{cases} z_n^{(i)} & k=0 \\ \nabla^{k-1} z_n^{(i)} - \nabla^{k-1} z_{n-1}^{(i)} & k > 0 \end{cases}\\
&\nabla_p^k z_n^{(i)} = \begin{cases} p_n^{(i)} & k=0 \\ \nabla_p^{k-1} z_n^{(i)} - \nabla^{k-1} z_{n-1}^{(i)} & k > 0 \end{cases}
\end{aligned}\right\} \tag{1.3}$$

The formulas which are used require backward differences $\nabla^k z_n^{(i)}$ only for $i=\ell$, $0 \le \ell \le d$, where the choice of ℓ determines the characteristics of the formula. The larger the value of ℓ the more accurate the formulas, but because of stability considerations values of $\ell < d$, are sometimes best. If $\ell < d$, the corrector equations given below require (in general) the solution of a nonlinear system of equations for $z_{n+1}, z'_{n+1}, \ldots, z^{(d-1)}_{n+1}$. The $\gamma_{k,j}$, and $\beta_{k,j}$ in the formulas are constants which can be determined by requiring that the formulas be exact for as high degree polynomial as possible.

The predictors are:

$$p_{n+1}^{(\ell-j)} = \sum_{i=0}^{j-1} \frac{h^i}{i!} z_n^{(\ell-j+i)} + h^j \sum_{k=0}^{q-1} \gamma_{k,j} \nabla^k z_n^{(\ell)} \qquad j=1,2,\ldots,\ell \qquad (1.4)$$

$$p_{n+1}^{(\ell+j)} = h^{-j} \sum_{k=j}^{q-1} \beta_{k,j} \nabla^k z_n^{(\ell)} \qquad j=0,1,\ldots,d-\ell-1 \qquad (1.5)$$

The correctors are:

$$z_{n+1}^{(\ell-j)} = p_{n+1}^{(\ell-j)} + h^j \gamma_{q,j} \nabla^q p_{n+1}^{(\ell)} \qquad j=1,2,\ldots,\ell \qquad (1.6)$$

$$z_{n+1}^{(\ell+j)} = p_{n+1}^{(\ell+j)} + h^{-j} \beta_{q,j} \nabla^q z_{n+1}^{(\ell)} \qquad j=1,2,\ldots,d-\ell-1 \qquad (1.7)$$

$$f\left(t_{n+1}, z_{n+1}, \ldots, z_{n+1}^{(d-1)}\right) = h^{\ell-d} \sum_{k=d-\ell}^{q-1} \beta_{k,d-\ell} \nabla^k z_n^{(\ell)} + h^{\ell-d} \beta_{q,d-\ell} \nabla^q z_{n+1}^{(\ell)} \qquad (1.8)$$

If ℓ=d formulas (1.5), (1.7), and (1.8) are not used. If ℓ=d-1, formula (1.7) is not used. If $\ell < d$, $\nabla_p^q z_{n+1}^{(\ell)}$ in formula (1.6) is replaced by $\nabla^q z_{n+1}^{(\ell)}$. Finally if $z^{(d)}$ in eq. (1.2) is replaced with zero, then the right hand side of formula (1.8) is replaced with zero. Note that in this last case one must have $\ell < d$.

II. THE SUBROUTINES

A. The Core Integrator -- DVØA

The core integrator has the following jobs.

1. Keeping track of the various ways the user can request output points, and returning control to the user at these points with information of interest to the user set to the proper values.
2. Keeping the past history of the solution, which is stored in difference tables, up to date.
3. Predicting the new value of the dependent variables on each new step. (Formulas (1.4) and (1.5).)
4. Correcting the values of those dependent variables which are not being treated as stiff equations. (Formula (1.6), if ℓ=d. If $\ell \neq$d, all corrections are carried out in DSTF.)
5. Estimating the error with the current stepsize.
6. Estimating what the error would be if the stepsize were doubled.
7. Determining if the user has requested more accuracy than the precision of the computed derivatives warrants.
8. Selecting the order of integration formula to be used on each equation in the system of differential equations. (The order is selected independently for each equation in the system.)
9. At the option of the user, to provide output which enables one to see why the integrator is selecting the stepsize and the integration orders the way it is.
10. Selecting the stepsize. (That is,deciding when to halve and when to double the mesh.)
11. Changing the stepsize, and the difference tables to correspond to the new stepsize.
12. On user option, to select the value of ℓ in formulas (1.4) to (1.8) when the current value of ℓ=d. The user must provide

the initial value of ℓ. (This option will only be made available if we can find a simple reliable way of providing it.)

The SUBRØUTINE statement has the form

```
SUBRØUTINE DVØA(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL,
          MXSTEP,KSTEP,KEMAX,EMAX,KQ,YN,DT)
```

One calls DVØA to set up initial communication and to initialize certain variables internal to DVØA. Other entries and their function are given below.

```
ENTRY DVØAR
```

DVØAR is called to restart an integration, while maintaining a distance of DELT between the output points that one gets with IFLAG=3.

```
ENTRY DVØAØ(LEX,LSS,LØØ,LHØ)
```

DVØAØ is called to set up certain special returns, and optional output.

```
ENTRY DVØAS(KS,ASTIF,MCHANG,STIFAC,SGAM)
```

DVØAS is called by the subroutine DSTF, to set up communication between DVØA and DSTF. The user who doesn't want to set up code for stiff equations can call DVØAS in order to find out if DVØA finds some of his equations to be stiff.

```
ENTRY DVØAV(NVE,EPSV)
```

DVØAV is called to set up DVØA to handle variational equations.

```
ENTRY DVØAH(HMAXA,LHC)
```

DVØAH is called to specify a maximum stepsize, or to specify when the stepsize is to be halved or doubled.

```
ENTRY DVØAG(IG)
```

DVØAG is called by DAGS to take care of the various disruptions that occur when locating GSTOPs.

```
ENTRY DVØA1
```

DVØA1 is called by the user whenever he wants to get back to the integrator and simply continue with the integration.

B. The Interpolator -- DAINT

The interpolator computes the values of the dependent variables and their derivatives (up to the order of the differential equation) at arbitrary values of the independent variable.

The SUBRØUTINE statement has the form

```
SUBRØUTINE DAINT(T,Y,F,KD,KQ,YN,DT,NEV,NY,TL,H)
```

One calls DAINT to set up initial communication. This set up call is made by DVØA if one is doing an integration.

```
ENTRY DAINT1
```

DAINT1 is called whenever one wants interpolated values to be computed.

C. The GSTOP Locator -- DAGS

The GSTOP locator finds the zeros of arbitrary functions $g_j(t,\vec{y})$, where t and $\vec{y}$ are as defined in eq. (1.1).

The SUBRØUTINE statement has the form

```
SUBRØUTINE DAGS(T,Y,IFLAG,NG,NSTØP,LTGS,G,GT)
```

One calls DAGS to set up initial communication.

```
ENTRY DAGS1
```

DAGS1 is called to check for zero crossings, and also in the process of locating the zeros.

D. The CHECK Subroutine -- DCHK

The CHECK subroutine gives a simple way for the user to get labeled output of everything in the calling sequence of DVØA. Such output is frequently useful in debugging a program.

The SUBRØUTINE statement has the form

```
SUBRØUTINE DCHK(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL,
          MXSTEP,KSTEP,KEMAX,EMAX,KQ,YN,DT,IØ)
```

One calls DCHK to set up initial communication.

ENTRY DCHK1

DCHK1 is called whenever one wants to print out the values of the variables in the calling sequence of DVØA.

E. The Stiff Integrator -- DSTF

The stiff integrator has the following jobs.

1. Form the matrix used in the iteration for solving eqs. (1.6)-(1.8). This matrix is described in subsection IV E below.
2. Deciding when to compute the Jacobian matrix.
3. Carrying out the iterations used to solve eqs. (1.6)-(1.8).
4. At the option of the user, to provide output of internal variables associated with the solution of eqs. (1.6)-(1.8).
5. On user option, to select the value of ℓ used in eqs. (1.4)-(1.8), when $\ell < d$.
6. Changing the difference table from one value of ℓ to the corresponding table for a different value of ℓ.
7. Obtaining an estimate of what accuracy can be achieved in solving eqs. (1.6)-(1.8). (This will only be done if it is necessary for job 7 in the core integrator.)
8. Determining if the user has computed the Jacobian incorrectly.

The SUBRØUTINE statement has the form

```
SUBRØUTINE DSTF(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL,
              MXSTEP,KSTEP,KEMAX,EMAX,KQ,YN,DT,KS,ASTIF,JSTØR,
              FPS,IFPS,WS,IWS,IW)
```

One calls DSTF to set up initial communication with DSTF and DVØA, and to initialize certain internal variables in these subroutines. Other entries and their function are given below.

ENTRY DSTFR

DSTFR is used to restart an integration using DSTF in exactly the same way as DVØAR is used with DVØA.

ENTRY DSTFØ(LMF,LØI)

DSTFØ is called to set up a special return just before the matrix used in the iteration is factored, and to set up optional output.

ENTRY DSTFC

DSTFC is called when the user wants to change the values of ℓ used in eqs. (1.4)-(1.8).

ENTRY DSTFJ

DSTFJ is called if one wants to use a new Jacobian matrix in the iteration to solve eqs. (1.6)-(1.8) without the Jacobian calculation being requested by DSTF.

ENTRY DSTFV(NVE,EPSV,IREF)

DSTFV is called to set up DSTF to handle variational equations.

F. The Partial Derivative Generator -- DPART

The partial derivative generator is used to generate the Jacobian matrix required by the stiff integrator. It is used when the user doesn't want to (or can't) write the code for the analytic partial derivatives. More work is required before specifications for this subroutine can be given.

III. USAGE

A. The Basic Framework

In this subsection we describe how to use the subroutines to integrate the basic initial-value problem (1.1). We begin by listing all of the parameters in the call to the integrator together with how they are used. This is followed by a sample FORTRAN-like program which can be used as a guide when writing a program which calls the integrator. In the sample program we use symbols for statement numbers. Some of these symbols are referred to later in connection with other options.

NEQ — number of differential equations in the system. (Same as m in eq. (1.1).)

T — independent variable. (Same as t in eq. (1.1).)

Y — vector of dependent variables. (Same as $\vec{y}$ in eq. (1.1).) For a system of first order equations Y(I) is the I-th dependent variable. In the case of higher order equations, Y(I) is the same as the I-th component of $\vec{y}$ as indicated just below eq. (1.1). Thus for the system F(1) = U", F(2) = V"; Y(1) = U, Y(2) = U', Y(3) = V, Y(4) = V'.

F — vector of derivative values. (F(I) is the same as f_I in eq. (1.1).) The user must provide code to compute F given T and Y.

KD — order of the differential equations in the system. Thus for a system of first order equations, KD=1; for a system of second order equations, KD=2; etc. If differential equations of mixed order are to be integrated, KD must be a vector and KD(1) < 0 to inform the integrator that this is the case. The order of the I-th equation is then given by |KD(I)|. (Thus |KD(I)|=d_I in eq. (1.1); for I > 1, KD(I) may be either positive or negative.) Differential equations

with order greater than 4 can only be integrated by breaking them into lower order equations, or by changing certain DATA statements in the subroutines.

EP — bound on the estimated local error. The local error in integrating each differential equation is kept less than EP/10 unless noise appears to be limiting the precision. If noise appears to be limiting the precision then the local error estimate is permitted to exceed EP/10; for further details on this, see the usage of EMAX and IFLAG=6. For different error bounds on different equations, let EP(I) < 0 for I < K, and let EP(K) ≥ 0. The local error control for the I-th equation is then based on |EP(I)| for I < K and on EP(K) for I ≥ K. For differential equations of order d > 1, the error estimate is for the error in the (d-1)-st derivative. In this case the value of EP necessary to maintain a given accuracy in all lower order derivatives depends on the scaling. For a relative error bound, see subsection III B below. EP = 0 is not permitted except when HMAXA = 0. For further details on this see subsection III G below.

IFLAG — parameter used for communication between the integrator and the user. The integrator sets IFLAG as follows:

=1 The value of Y for the current step has been predicted. The user should compute F and call DVØA1.

=2 The value of Y for the current step has been corrected. The user should compute F and call DVØA1.

=3 An output point has been reached (see the usage of DELT), to continue the integration call DVØA1.

=4 T=TFINAL. To continue the integration with a different TFINAL change TFINAL and call DVØA1. If DVØA1 is called without changing TFINAL, IFLAG is set equal to 8.

=5 KSTEP ≥ KSØUT. (See the description of MXSTEP.)

=6 EMAX > .1 and it appears that reducing H will not help reduce the global error because of round-off error or noise in computing F. A larger value of EP (or of |EP(KEMAX)| if EP is a vector) should probably be used. If EP is not increased, too small a stepsize is liable to be used. We have found that replacing EP with 32*EMAX*EP works reasonably well. (Note that in most cases, EMAX will be only slightly larger than .1.) Increasing EP in this way should not degrade the accuracy; however, if the nature of the problem changes it may pay to use a smaller value of EP later in the integration.

=7 |H| < HMINA. This may be the result of halving H, of coming to the end of the starting phase with |H| < HMINA, or of the user increasing the value of HMINA. If one wishes to continue with the current value of H, set HMINA ≤ H and call DVØA1. If H has just been halved (in which case EMAX > .1), one may continue with the old stepsize by simply calling DVØA1. (Such action is risky without a careful analysis of the situation.) If the stepsize has not just been halved, calling DVØA1 will result in the integration continuing with the current value of H, and a return to the user with IFLAG=7 will occur at the end of every step until |H| ≥ HMINA.

=8 Fatal error. The value of KEMAX gives the source of the error as indicated below. If DVØA1 is called an error message which includes the value of KEMAX will be printed, and the program stopped.

KEMAX =1 NEQ ≤ 0.

=2 H=0

=3 KD (or some |KD(I)|) is equal to 0 or greater than 4.

=4 DELT ≈ 0 (After IFLAG=3, Old TØUT=(Old TØUT)+DELT.)

=5 TFINAL was not changed after IFLAG=4.

=6 EP (or some EP(I))=0.

=7 At the initial point the direction of integration is in the direction opposite from TFINAL, and the estimated error in extrapolating back to TFINAL from the initial point is too large.

=8 During the integration, either TFINAL or DELT has been specified in such a way that the next output point, TMARK, satisfies (TL-TMARK)/H > 8 where TL is the last value of T that the integration has reached. (Thus the output point specified is one that the integration process passed over 8 steps ago, assuming that H hasn't changed.) If one wants to continue under such conditions, set IFLAG=0, call DVØA1, and Y and F will be computed at this output point and a return will be made with IFLAG=3 or 4 as usual. Otherwise a call to DVØA1 results in the program being stopped as indicated above.

=9 DVØA1 was called immediately after a call to DVØAV or DVØAS. After a call to DVØAV or DVØAS, either DVØA or DVØAR must be called.

=10 NVE is specified improperly (possible only when the entry for variational equations is used).

=11 DVØAH was called with LHC≠0 and IFLAG≠9.

=12 After a return from DVØA, DAGS1 was called with IFLAG equal to something other than 1, 2, 4, or 9.

=13 DAGS1 was called after DAGS was called with NG ≤ 0.

The following errors are possible only with stiff equations.

KEMAX=14 MØD(|KS(I)|,10) > 4 or KS(I) = -k*10 where k is a positive integer.

=15 JSTØR specified improperly.

=16 IFPS is too small.

=17 IWS is too small.

=18 DSTF1 was called immediately after DVØA set IFLAG equal to something other than 1, 2, or 10.

=19 DSTF1 was called immediately after a call to DSTFV. After a call to DSTFV either DSTF or DSTFR must be called.

=20 IREF (in DSTFV entry) is specified improperly.

=21 The value of KS for a variational equation does not agree with the value of KS for the equation with which the variational equation is associated

H the stepsize. The initial value of H determines the direction of integration. The following points should be considered in selecting the initial value of H.

1. Later values of $H=2^k \cdot$(initial H) where k is an integer.
2. Efficiency does not depend critically on the initial value of H, see for example Table 12 in [2].
3. If it does not lead to problems in computing the derivatives (e.g. because of overflow or because of trying to compute the square root of a negative number), it is better to choose H much too large than too small.
4. If one wants to obtain output values without the integrator doing an interpolation, then one must choose H and DELT such that $DELT=H \cdot 2^k$, k a non-negative integer. (In addition DELT must be an integer times a power of 2 since otherwise there is liable to be a problem with round-off error. Thus $DELT=2^{-3}$, $3x2^{-5}$, $13x2^{-7}$, $51x10^{-9}$ are all satisfactory, but DELT=.1 will always lead to a need to perform interpolations.)

HMINA minimum stepsize permitted after the integration is started. The user cannot restrict the minimum value of $|H|$ while the integration is getting started. After the integration is started, and whenever H is halved, $|H|$ is compared with HMINA. If $|H| <$ HMINA control is returned to the user with IFLAG=7.

DELT output increment. Initially the integrator sets the internal variable TØUT equal to the initial value of T.

Whenever T=TØUT a return is made to the user with IFLAG=3. If DVØA1 is called after IFLAG=3, TØUT is replaced with T+DELT. If TØUT does not fall on an integration step, interpolated values for Y and F are computed on the first step that (T-TØUT)*H > 0 and T is set equal to TØUT. If one selects an initial value of DELT such that H*DELT < 0, the sign of DELT is changed by the integrator and the initial value of TØUT is set equal to T+DELT. (Thus if H*DELT < 0 initially, one does not get output at the initial point.)

TFINAL — final value of T. When T reaches TFINAL, control is returned to the user with IFLAG=4. Output values at TFINAL are obtained in the same way as described for TØUT in the description of DELT above.

MXSTEP — maximum number of steps. Initially the integrator sets KSTEP=0 and the internal variable KSØUT = |MXSTEP|. KSTEP is increased by 1 at the end of each step. Whenever KSTEP ≥ KSØUT, a return to the user is made with IFLAG=5. If DVØA1 is called after IFLAG is set equal to 5, KSØUT is replaced by KSTEP+|MXSTEP|. If MXSTEP < 0, KSØUT is also replaced by KSTEP+|MXSTEP| after IFLAG is set equal to 3 or 4. Thus if MXSTEP < 0, |MXSTEP| is the maximum number of steps that can occur before IFLAG = 3, 4 or 5.

KSTEP — number of integration steps taken. KSTEP is initialized and incremented by the integrator.

KEMAX — index of the equation responsible for EMAX (see below). KEMAX also indicates the type of error when IFLAG=8 (see above).

EMAX — largest value in any equation of (estimated error)/ε where ε = |EP| for the equation under consideration. Ordinarily the stepsize is halved if EMAX > .1. However, with a recent history of round-off error or noise in the derivative evaluations limiting the precision, values

of EMAX as large as 1 are permitted. If round-off or noise appears to be limiting the precision on the current step, the stepsize is not halved under any circumstances.

KQ — vector used to store integration orders. KQ(I) gives the value of q for the I-th equation. See equations (1.4) and (1.6) for the definition of q(ℓ=d).

YN — vector used to store Y at the end of each integration step.

DT — a two dimensional array used to store the difference tables. DT(I,J)=$\nabla^{I-1}F_n(J)$ where $\nabla^0F_n(J)=F_n(J)$= value of the J-th component of F on the n-th step, and $\nabla^{k+1}F_n(J)=\nabla^kF_n(J)-\nabla^kF_{n-1}(J)$, k=1,2,...,KQ(J).

The sample program is given below.

In the type and dimension information given below, (?) indicates that the variable may be either a scalar or vector (see description above), m ≥ NEQ, and n ≥ total order of the system.

```
INTEGER           NEQ,KD(?),IFLAG,MXSTEP,KSTEP,KEMAX,KQ(m)
REAL              EP(?),HMINA,EMAX
DOUBLE PRECISION  F(m),DT(20,m)
DOUBLE PRECISION  T,Y(n),H,DELT,TFINAL,YN(n)
```

A_0 — Assign the values for NEQ,KD,MXSTEP,EP,HMINA,DELT, and TFINAL. Assign initial values for T, Y, and H. (These will be updated by the integrator.) Values need not be assigned for IFLAG,KSTEP,KEMAX,KQ,EMAX,F,DT, and YN.

A_1 — If certain special options are used they should be set up at this point.

```
CALL DVØA(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL,
     MXSTEP,KSTEP,KEMAX,EMAX,KQ,YN,DT)
```

GØ TØ A_3

A_2 CALL DVØA1

A_3 GØ TØ (I_1,I_2,...,I_8), IFLAG

I_1 CØNTINUE (Special code is required here if certain options are used.)

I_2 Compute F(I), I=1,2,...,NEQ (F=F(T,Y))
GØ TØ A_2

I_3 Print results (T=TØUT=Last TØUT+DELT. See description of DELT.) Change DELT if desired (or STØP if |DELT| was set large)
GØ TØ A_2

I_4 Print results (T=TFINAL)
STØP or Change TFINAL and
GØ TØ A_2

I_5 Print results (KSTEP ≥ KSØUT See the description of MXSTEP.)
GØ TØ A_2 or STØP (Depending on how you wish to use MXSTEP.)

I_6 Increase EP as suggested under IFLAG=6 above.
Print new EP, T, etc.
GØ TØ A_2

I_7 Print "H IS TØØ SMALL"
STØP

I_8 GØ TØ A_2 (There IFLAG and KEMAX will be printed and the program stopped.)

Note that the actions taken in the above program from statement I_3 to the end are but suggestions of what one might do for the specified values of IFLAG. Many other things can be done.

The above basic framework may be added to as indicated in the options which follow.

B. Use of a Relative Error Test

Users frequently desire to bound the local error by $|\epsilon \cdot E(T,Y)|$ where ϵ is a constant and E is some specified function of T and Y. This option is obtained in the program given in subsection A above with the following changes. At A_0 assign a value to ϵ instead of to EP; at I_1 set EP=$\epsilon \cdot E(T,Y)$ before continuing to I_2; and at I_6 increase the value of ϵ instead of the value of EP. Either ϵ or E can be a vector in which case EP is a vector, and the values stored in EP must be negative as mentioned in the description of EP previously. Users frequently set E equal to the vector $(|Y(1)|, |Y(2)|,\ldots,|Y(NEQ)|)$ (when KD=1) in order that the local error requested be relative to the solution. Unfortunately, this is frequently a poor choice.

C. Restarting an Integration

Because of discontinuities, or a change in the differential equations being integrated, it is sometimes best to restart the integration. This of course can be done by making the appropriate adjustments, calling DVØA and then going to A_3. This procedure, however, is liable to change the spacing one gets on the output when IFLAG=3 because of the way TØUT is initialized, as explained in the description of DELT. To preserve the spacing simply substitute

```
CALL DVØAR
```

for the CALL DVØA(NEQ,...), and then go to A_3.

D. Special Returns and Optional Output

At any place in the program one may insert

```
CALL DVØAØ(LEX,LSS,LØØ,LHØ)
```

The variables LEX, LSS, LØØ, LHØ are all type INTEGER; and each refers to a special option as indicated below. Any option one is not interested in will be left unchanged if the corresponding variable is equal to 0. The integrator does not change the value of any of these variables, and any time the user changes one of them DVØAØ (or DVØA or DVØAR) must be called for the change to take effect.

LEX determines whether output at TFINAL is to be obtained using extrapolation or interpolation. Extrapolation is useful if F has a singularity at TFINAL.

>0 Obtain values with extrapolation. (If TFINAL falls on an integration step, the extrapolation occurs before the step is taken.)

=0 No change. (The nominal internal flag is set for interpolation.)

<0 Obtain values with interpolation.

LSS is used to specify whether one wants the integrator to return control at the end of each step. Control is returned with IFLAG=9. Thus if this option is used one must add an I_9 in statement A_3, and then add a statement numbered I_9 where one does whatever he wants to do at the end of each step, followed by a GØ TØ A_2.

>0 Return control to the user with IFLAG=9 at the end of every step.

=0 No change. (The nominal internal flag is set for no return.)

<0 No output.

LØØ is used to specify that the integrator is to print certain intermediate results on every step. This information is useful for debugging and in analyzing the error control that is used. The output to be provided is not yet completely specified.

>0 Give the output.

=0 No change. (The nominal internal flag is set for no output.)

<0 No output.

LHØ is used to specify whether one wants control returned when the stepsize is changed. Control is returned with IFLAG=11 just before the stepsize is halved and with IFLAG=12 just before the stepsize is doubled. Thus if this option is used one must provide for larger values of IFLAG in statement A_3. No special returns are made if the stepsize is reduced in the process of

taking the first step. Control is returned with the current value of Y contained in YN and the current value of F(I) in DT(1,I). When the stepsize is halved Y and F contain the values that resulted in the decision to halve the step. To continue, call DVØA1.

>2 Control is returned just before the stepsize is changed.

=2 Control is returned just before the stepsize is doubled.

=1 Control is returned just before the stepsize is halved.

=0 No change. (The nominal internal flag is set for no return when H is changed.)

<0 Control is not returned when the stepsize is changed.

E. Variational Equations (and Others With Similar Characteristics)

Consider the system of differential equations (compare with eq. (1.1))

$$\begin{aligned} z_i^{(d_i)} &= f_i(t,\vec{y}) \quad i=1,2,\ldots,m \\ &\qquad\qquad \vec{y}_A(t_0) = \vec{\eta}_A, \qquad (3E.1) \\ z_i^{(d_i)} &= f_i(t,\vec{y}_A) \quad i=m+1,\ldots,m+\mu \end{aligned}$$

where all variables common to eq. (1.1) are defined the same as there, and $\vec{y}_A = (\vec{y},\vec{y}_a)$ where $\vec{y}_a = (z_{m+1}, z'_{m+1}, \ldots, z_{m+1}^{(d_{m+1}-1)}, z_{m+2}, \ldots, z_{m+\mu}^{(d_{m+\mu}-1)})^T$. Thus $\vec{y}_A$ is formed by augmenting the vector $\vec{y}$ with $\vec{y}_a$, the variables introduced in the bottom equation of (3E.1). If there are subexpressions in f_i, $i > m$, which depend only on t

and $\vec{y}$ then the option described here permits one to compute these subexpressions only once per step instead of the twice per step that would ordinarily be required. In addition accuracy is slightly better since only corrected values of $\vec{y}$ are used in computing the subexpressions. A common situation where this option can save significant time is in the integration of variational equations, in which case the bottom equation in (3.1) has the form

$$\vec{\zeta}_a = A(t,\vec{y})\vec{y}_a + \vec{b}(t,\vec{y}) \tag{3E.2}$$

where $\vec{\zeta}_a = (z_{m+1}^{(d_{m+1})},\ldots,z_{m+\mu}^{(d_{m+\mu})})^T$ and $A(t,\vec{y})$ is a block diagonal matrix of the form

$$A(t,\vec{y}) = \begin{pmatrix} \alpha(t,\vec{y}) & & 0 \\ & \alpha(t,\vec{y}) & \\ & & \ddots \\ 0 & & \alpha(t,\vec{y}) \end{pmatrix}, \quad \alpha(t,\vec{y}) = \begin{pmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial y_1} & \cdots & \frac{\partial f_m}{\partial y_n} \end{pmatrix}, \tag{3E.3}$$

where y_i is the i-th component of $\vec{y}$ and $n = \sum_{i=1}^{m} d_i$. Also $d_j = d_i$ if $j \equiv i \bmod m$. One might have for example $y_{i+kn} = \partial y_i/\partial \eta_k$ $i,k = 1,2,\ldots,n$, where η_k is the initial condition for y_k. In this case the first mn components of $\vec{b}$ are 0. If $f_i(t,\vec{y})$ can be written as $f_i(t,\vec{y},\beta_1,\ldots,\beta_j)$ where the β's are parameters which are fixed on a given integration and one wants to compute $\partial y_i/\partial \beta_k$ in addition to the $\partial y_i/\partial \eta_k$ computed as indicated above, simply set $y_{n^2+i+kn} = \partial y_i/\partial \beta_k$ and $b_{mn+i+(k-1)m} = \partial f_i/\partial \beta_k$,

where b_ν is the ν-th component ov $\vec{b}$. The matrix A then has $m(n+j)$ rows and $n(n+j)$ columns, with the block diagonal structure as indicated in (3E.3). In the case of variational equations this option cuts in half the number of evaluations of $\alpha(t,\vec{y})$ and $\vec{b}(t,\vec{y})$, a considerable savings in some cases.

This option requires the definition of two new parameters.

NVE — is the number of additional equations. (NVE = μ and NEQ = m in (3.1) above. The total number of equations is given by NEQ+NVE.)

EPSV — is an error tolerance for the variational equations. If EPSV > 0, EPSV is used for EP in all equations with index > NEQ. If EPSV = 0 no error estimates are computed for those equations with index > NEQ, and the step-size will be selected on the basis of the estimated errors in the first NEQ equations. If EPSV < 0, then $|EP(I)|$, (I > NEQ) will be used in the error control on the I-th equation where, as before, if EP(I) ≥ 0 for some I, this value will be used for all larger I. (Note that if EPSV > 0 and one is using a relative error test as described in subsection B, then EPSV must be changed in addition to EP when IFLAG=1.)

As implied in the above, Y is redefined now as the augmented vector $\vec{y}_A$ in eq. (3E.1) and F is extended to contain f_i, i=1,2,...,NEQ+NVE.

To use this option the following changes should be made in the sample program given in subsection A above.

In the dimension information one must have m ≥ NEQ+NVE, and n ≥ total order of the system including the equations which have been added. If one has mixed orders, the dimension of KD must be ≥ m, otherwise KD can be a scalar. One should add NVE to the variables specified to be of type INTEGER, and add EPSV to those specified to be of type REAL.

At A_0 initial values for NVE and EPSV must be assigned.

At A_1 (before the call to DVØA)

```
CALL DVØAV(NVE,EPSV)
```

Code from A_3 to the GØ TØ A_2 just below I_2 should be replaced with:

A_3 GØ TØ ($I_1, I_2, \ldots, I_{10}$), IFLAG

I_1 CØNTINUE (Special code is required here if certain options are used.)

V_1 Compute F(I), I=1,2,...,NEQ
GØ TØ A_2

I_{10} Compute subexpressions for extra equations. ($\alpha(t,\vec{y})$ and $\vec{b}(t,\vec{y})$ in the case of variational equations.)

I_2 Compute F(I), I=NEQ+1, NEQ+2, ..., NEQ+NVE
(Note that if the F(I) computed at I_2 depend only on $\vec{y}$ (thus not on $\vec{y}_a$) in eq. 3E.1, then these F(I) need not be computed when IFLAG=2, since the values will be the same as those obtained when IFLAG=10.)
IF(IFLAG-2) any, V_1, A_2 (IFLAG < 2 should be impossible.)

If one is not using the end of step return (LSS) in subsection D, simply set $I_9=A_2$.

We have introduced the feature described in this section in its simplest form for convenience in its description. It is actually available for the more general case

$$
\begin{aligned}
z_i^{(d_i)} &= f_i(t,\vec{y}) && i=1,2,\ldots,m,\\
z_i^{(d_i)} &= f_i(t,\vec{y}_A) && i=m+1,\ldots,m+\mu_1,\\
z_i^{(d_i)} &= f_i(t,\vec{y}_B) && i=m+\mu_1+1,\ldots,m+\mu_1+\mu_2,\\
z_i^{(d_i)} &= f_i(t,\vec{y}_C) && i=m+\mu_1+\mu_2+1,\ldots,m+\mu_1+\mu_2+\mu_3,
\end{aligned}
\qquad \vec{y}_X(t_0)=\vec{\eta}_X \qquad (3E.4)
$$

where $\vec{y}_A = (\vec{y},\vec{y}_a)$ as before, $\vec{y}_B = (\vec{y}_A,\vec{y}_b)$ where $\vec{y}_b$ gives the the variables introduced in the third set of equations, $\vec{y}_C = (\vec{y}_B,\vec{y}_c)$, etc. To get this extended capability, NVE and EPSV are used as follows.

NVE is a vector, with

NVE(1) = $-\nu$ where ν is the number of additional sets of equations, and NVE(1) < 0 is necessary to indicate that NVE is a vector. Thus if one has the system given in eq. (3E.4) down to and including $z_i^{(d_i)} = f_i(t,y_C)$, then ν=3 and NVE(1) = -3, and the total number of equations is $m+\mu_1+\mu_2+\mu_3$.

NVE(2) is used to tell the user what set of equations requires a derivative evaluation when IFLAG=10. If we let μ_0 = m=NEQ, then when IFLAG=10 and NVE(2)=j, F(I) is to be computed for I = $\mu_0+\mu_1+\cdots+\mu_{j-1}+1,\ldots,\mu_0+\mu_1+\cdots+\mu_j$.

NVE(2+k) is the number of equations in the k-th additional set ($=\mu_k$), k=1,2,...,ν. The total number of equations is thus given by NEQ+NVE(3)+···+NVE(2+ν) where ν = -NVE(1).

EPSV is a vector, with

EPSV(1) used to flag the usage of EPSV as follows

> 0	ϵ_j = EPSV(2)
= 0	ϵ_j = 0 (In this case EPSV can be a scalar.)
< 0	ϵ_j = EPSV(j+1)

where

$\epsilon_j > 0$ means use ϵ_j for the error tolerance in every equation of the j-th additional set. ($I=\mu_0+\cdots+\mu_{j-1}+1,\ldots,\mu_0+\cdots+\mu_j$).

$\epsilon_j = 0$ means do not check the error in the j-th additional set.

$\epsilon_j < 0$ means refer to EP(I) (passed through the call to DVØA) for the error tolerance, $I=\mu_0+\cdots+\mu_{j-1}+1,\ldots,\mu_0+\cdots+\mu_j$. (Of course, if EP(I) ≥ 0 for some I in this range, that value will be used for all larger values of I in the j-th set of additional equations.)

Except for the obvious changes, usage is identical with that given earlier for the case when NVE is a scalar down to the statement labeled I_{10} (see the middle of page 24). From there on one should use the following.

I_{10} Compute subexpressions for the j-th additional set, where j=NVE(2).
Compute derivatives for the j-th additional set.
GØ TØ A_2.

I_2 Compute F(I), for all I > NEQ. (As before, one should not bother to compute those F(I) whose values have not changed since they were computed when IFLAG=10.)
GØ TØ V_1.

F. Direct Interpolation From Results, Present and Past

During an integration one can obtain values of Y and F at any time for any value of T by simply setting T to the desired value and then executing

```
CALL DAINT1
```

However, if one uses this option when IFLAG is not equal to 3, 4, 5, 6, or 9, then the values of T, Y and F must be saved and restored if one intends to continue the integration. Note also that one can expect reasonable accuracy in the interpolation only if

$$-q_{min} \leq (T-TL)/H \leq 1 \qquad (3F.1)$$

where TL is the value of T at which the difference tables DT were computed, H is the stepsize, and $q_{min} = \min\{KQ(I)\}$.

To save the solution for retrieval at a later time the following changes should be made in the sample program of subsection A.

Just below A_0 set an integer variable KSTØRE = -1, at A_1 insert

```
CALL DVØAØ(LEX,LSS,LØØ,LHØ)
```

where LSS > 0, LHØ=1 (or is ≥ 3) and subject to these constraints one has set the parameters passed in this call to appropriate values depending on what one wants. (See subsection III D.)

At A_3 add (at least) I_9, I_{10}, and I_{11} to the computed GØ TØ. (If there are no variational equations, set $I_{10}=A_2$.) At I_9 the following code should be inserted.

```
I9      Do any other thing you may want to do when IFLAG=9.
        KQMIN=100
        DØ S1 I=1,NEQ (Or NEQ+NVE if variational equations are
        present.)

S1      KQMIN=MINO(KQMIN,KQ(I))
        IF (KSTEP.LT.(KSTØRE+KQMIN)) GØ TØ A2

I11     Store T,H,YN,KQ,DT (on tape, disc, or drum)
        KSTØRE=KSTEP
        Make special test for IFLAG=11 and transfer if you want
        to distinguish between IFLAG=11 and IFLAG=9.
        GØ TØ A2
```

To retrieve the solution at a later time, one first assigns values to TL,H,YN,KQ, and DT, from the values stored earlier, where TL is one of the stored values of T, and the remaining variables were stored at the same time as TL. The value of TL retrieved should satisfy (3F.1), where T is the value of the independent variable where values of Y and F are desired. In addition to variables already defined, one must assign values to:

NEV which is the total number of equations (=NEQ if there are no variational equations), and

NY which is the total number of components in Y (=the total order of the system)

Before the first interpolation, one makes the set-up call

```
CALL DAINT(T,Y,F,KD,KQ,YN,DT,NEV,NY,TL,H)
```

(All of these variables have been defined above.) Then to obtain Y and F given T

```
CALL DAINT1
```

remembering that TL,H,YN,KQ, and DT are equal to previously stored values, and are selected as a function of T.

One may not use DAINT in this way at the same time as an integration using DVØA is being performed. One may of course, get such a capability by using another interpolation program identical to DAINT, except with the names DAINT and DAINT1 changed to something else. (One may also have to change the names of other variables to avoid conflict with those used in the integration.)

Of course, if one is only interested in interpolating for the variables associated with some of the equations, one need store only the corresponding values of YN,KQ, and DT. The obvious changes in KD,NEV, and NY should then be made when retrieving the information.

G. Getting More Direct Control Over the Integration

The options described here give the user several levels of more direct control. One can merely specify that the stepsize is not to exceed a certain maximum value, or one can force the integrator to halve or double the stepsize at specified times, or one can specify the integration orders to be

used in addition to specifying the stepsize. These options are provided for those masochistic users who like to have enough rope to hang themselves.

The following two parameters are used.

HMAXA determines an internal variable, HMAX, so that doubling H is not permitted if doing so would result in an H which satisfies $|H| > HMAXA$. (Normally HMAX is so large that it never limits the stepsize.) HMAXA is type REAL.

LHC determines the action to be taken as follows. LHC is type INTEGER.

- =0 Assigns the internal variable HMAX as specified by HMAXA.
- =-1 Same as LHC=0, except in addition the stepsize is halved.
- =-2 Same as LHC=0, except in addition the stepsize is doubled.
- =1 The stepsize is halved, and HMAX is set so that H will not be doubled unless the user so requests. (HMAXA is ignored.)
- ≥2 The stepsize is doubled and HMAX is set so that H will not be doubled again unless the user so requests. (HMAXA is ignored.)

One gets the desired action by means of the statement

```
CALL DVØAH(HMAXA,LHC)
```

This call can be made at any time if LHC=0. Otherwise it can only be made after IFLAG has been set equal to 9. Thus one must make use of the LSS option of subsection III D. If LHC≠0, one should call DVØAH only after everything else that one might want to do when IFLAG=9 has been done, and the call to DVØAH should be followed by a GØ TØ A_3 in the sample program of subsection A. If LHC=0, one should simply continue as usual after the call to DVØAH, If DVØAH is called with LHC≠0, and IFLAG was not previously set to 9, then after the return from the call to DVØAH, IFLAG will equal 8, a fatal error condition. In using the above options, one can insure that the stepsize will not be halved by setting EP

(or EP(1)) to a ridiculously large positive value, or by setting HMINA to an appropriate value whenever the stepsize is changed.

One can eliminate any error estimation, stepsize control, or integration order selection on the part of the integrator by calling DVØAH with HMAXA=0, and LHC ≤ 0. After this call, which sets the internal variable HMAX=0, if one sets EP (or EP(1))=0 one has complete control over the integration, and no changes will be made in KQ or H unless the user makes them. (The reason HMAX must be set to zero, is to protect those users who set EP=0 by accident.) If this option is used, one should not change KQ(I) by more than ±1 on any step. One should not use this option when the integration is being started.

Anytime the user takes over the control of H, we suggest that he get the output provided with LØØ of subsection III D on a sample problem, as a partial check on the validity of what he is doing.

H. Locating Zeros of Arbitrary Functions (GSTOPs)

A GSTOP is an indication to the user that a specified function, G(J), of the variables T and Y has a zero. The option described here is used to locate GSTOPs. Applications include the location of special output points, and the detection of singularities in F which must be found before an attempt to compute F is made.

Five new parameters are required for this option.

NG is the number of functions G(J) to be checked for zeros. If DAGS1 is called with NG ≤ 0, this is considered a fatal error, and IFLAG is set equal to 8.

NSTØP indicates the index of the function G with the zero when a GSTOP is found. Thus G(NSTØP) ≈ 0.

LTGS is a vector which indicates how the GSTOP is to be located.

LTGS(J)=0 means do not test for a zero in G(J). When equal to zero, LTGS(J) can be changed at any time, and LTGS(J)

can be set equal to zero at any time.

>0 means locate the zero using interpolation.

<0 means locate the zero using extrapolation. (Note that interpolation requires only half as many evaluations of G(J) as extrapolation, and gives much better control on the error. Thus LTGS(J) < 0 should only be used if extrapolation is a necessity.)

G is a vector of the functions whose zeros are to be located. The user supplies the code to compute G given T and Y.

GT is a vector used for temporary storage of past values of G.

To use this option, the following changes should be made in the sample program of subsection III A.

One should add the following to the type and dimension information, where k ≥ NG, and everything can be scalar if NG=1.

```
INTEGER           NG,NSTØP,LTGS(k)
DØUBLE PRECISIØN  G(k),GT(k)
```

At A_0 the values of NG and LTGS should be assigned.

At A_1 if one wants to test for GSTOPs from the beginning, or at any other place if one wants to start testing for GSTOPs at a later time, insert

```
CALL DAGS(T,Y,IFLAG,NG,NSTØP,LTGS,G,GT)
```

This call serves to set up communication. It should only be made again if the value of NG is changed.

The statement at A_3 is replaced with (The G_i are defined below.)

A_3 GØ TØ (X_1,X_2,I_3,I_4,I_5,I_6,I_7,I_8,X_3), IFLAG

where $X_1=X_2=G_4$ if LTGS(J) < 0 for any J,

and $X_1=I_1$, $X_2=I_2$ if LTGS(J) ≥ 0 for all J; and

$X_3=G_1$ if LTGS(J) > 0 for any J, and otherwise X_3 need only be included in the list of statement numbers at A_3 if DVØAØ (see subsection III D) is called with LSS > 0, in which case $X_3=I_9$. (Note that one must not call DVØAØ with LSS < 0 if LTGS(J) > 0 for some J. Also, if one has called DVØAØ with LEX > 0, then one should replace the I_4 in statement A_3 above, with G_2.)

Between statements labeled A_3 and I_1 insert the following. (It is understood that if LTGS(J) ≤ 0 for all J, then the statement at G_1 can be anything and similarly if LTGS ≥ 0 for all J, the statement at G_4 can be anything.)

G_1 Compute G(J) for all J which have LTGS(J) > 0

G_2 CALL DAGS1

G_3 GØ TØ (I_1,I_2,I_3,I_4,G_1,G_4,G_5,I_8,I_9,G_6), IFLAG

G_4 Compute G(J) for all J which have LTGS(J) < 0
GØ TØ G_2

I_9 Do anything special you want to do at the end of each step.
GØ TØ A_2

G_5 — The GSTOP has been found, G(NSTØP)≈0. Output results and/or whatever else you want to do. If one wants to change the definition of G at this time, one can do so by storing the values obtained with the new definition into the appropriate locations of GT. This is the only time the contents of GT should be altered.
GØ TØ A_2 or STØP or go someplace to restart the integration, etc.

G_6 G(NSTØP) changes sign, but DAGS could not get good convergence when iterating for the zero. This probably is due to a bug.
Print message indicating problem and
STØP or GØ TØ G_5

I. Extra Output for Checking Out a Program

When checking out a program it is frequently useful to see the values of all the variables involved in the integration. To set up the subroutine which provides labeled output of the integration variables, simply

```
CALL DCHK(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL,MXSTEP,
KSTEP,KEMAX,EMAX,KQ,YN,DT,IØ)
```

where all variables are the same as those in the call to DVØA, except for the additional variable IØ which is an INTEGER variable with usage as follows.

IØ =1 Print everything in the calling sequence
=0 Do not print anything
=-1 Print everything in the calling sequence except DT

If $|IØ| > 1$, then IØ is treated as a vector with dimension at least as large as NEQ, and is used as follows.

IØ(1)≥2 print all information associated with the first equation
IØ(1)=-2 same as IØ(1)≥2, except DT(K,1), k=1,2,... is not printed
IØ(1)≤-3 do not print variables associated with the first equation.

for I > 1 when $|IØ(1)| > 1$,

IØ(I)>0 print all information associated with the I-th equation
IØ(I)=0 do not print variables associated with the I-th equation
IØ(I)<0 same as IØ(I) > 0, except DT(K,I), K=1,2,..., is not printed

If one has variational equations and wants output for these also, then NTE should be substituted for NEQ in the call, where NTE is the total number of equations, and if IØ is a vector it should have dimension at least as large as NTE.

The output is obtained at any time by the simple statement CALL DCHK1

In most cases when this feature is used, one will also want to get the output available through the entry DVØAØ described in subsection III D.

J. Differential Equations With Side Conditions

In some applications, one is interested in solving eq. (1.1) with a side condition of the form

$$\vec{\varphi}(t,\vec{y}) = 0 \tag{3J.1}$$

where the true solution of eq. (1.1) satisfies (3J.1), and the vector $\vec{\varphi}$ has dimension $\leq$ the dimension of $\vec{y}$. The side condition (3J.1) is imposed to insure that the numerical solution of eq. (1.1) does not drift too far away from satisfying (3J.1). Such problems can be solved using the capability described in subsection III E above. Set NVE(1) = -1, and NVE(3) = 0, where the 0 indicates that there are no extra equations to be introduced and that the user would like a special return with IFLAG=10 at the time a return would ordinarily be made if extra equations were being introduced.

When IFLAG=10, the user is free to modify $\vec{y}$ as he wishes in order to satisfy (3J.1). In many cases a good solution to (3J.1) will be obtained by setting $\vec{y} = \vec{y}_c + \delta\vec{y}$, where $\vec{y}_c$ is the value of $\vec{y}$ returned by the integrator when IFLAG=10, $\delta\vec{y}$ is the vector of minimum length which satisfies

$$(\partial\vec{\varphi}/\partial\vec{y})\delta\vec{y} = -\vec{\varphi}(t,\vec{y}_c) \tag{3J.2}$$

and $(\partial\vec{\varphi}/\partial\vec{y})$ is the Jacobian matrix of $\vec{\varphi}$ with respect to $\vec{y}$.

IV. USAGE IN THE CASE OF STIFF EQUATIONS

In this section the stiff integrator is described following a format similar to that in Section III. To simplify the exposition we introduce

$\vec{u}$ = components of $\vec{z}$ in eq. (1.1) for which $\ell \equiv d$ in eqs. (1.4)-(1.8)

$\vec{v}$ = components of $\vec{z}$ in eq. (1.1) for which $\ell = d$, but which may have $\ell < d$ at some time in computation.

$\vec{w}$ = components of $\vec{z}$ in eq. (1.1) for which $\ell < d$ (4.1)

$\vec{x}$ = components of $\vec{z}$ in eq. (1.1) for which it is possible to have $\ell < d$ at some time in the computation.

The corresponding components of $\vec{y}$ and $\vec{f}$ are denoted by $\vec{y}_u, \vec{y}_v, \vec{y}_w, \vec{y}_x$, $\vec{f}_u, \vec{f}_v, \vec{f}_w$, and $\vec{f}_x$. In all cases the variables which make up the individual components of these vectors are in the same order as they are in $\vec{z}$, $\vec{y}$, and $\vec{f}$, respectively. In the usage below, the computations of $\vec{f}_v$ and $\vec{f}_w$ are placed together, since this simplifies the logic for the user. Better results would be obtained by only computing $\vec{f}_w$ at these places, and computing $\vec{f}_v$ at those places $\vec{f}_u$ is computed. It is also permitted to compute $\vec{f}_u$ where $\vec{f}_v$ and $\vec{f}_w$ are computed and to skip the computation of $\vec{f}_u$ at the places where the evaluation of $\vec{f}_u$ is indicated; although, of course, one will not get as good results doing this.

A. The Basic Framework

The stiff integrator requires all of the parameters listed in Subsection III A together with those listed below. Usage of the parameters given in III A is the same as indicated there, except for certain additions noted below.

IFLAG — After DVØA1 is called, usage is the same as in III A except when IFLAG=1 or 2 in which case one proceeds as follows.

=1 Y has been predicted. Compute $\vec{f}_v$ and $\vec{f}_w$ (see eq. (4.1)), and then call DSTF1.

=2 Y has been corrected. Compute $\vec{f}_v$ and $\vec{f}_w$ and call DSTF1.

After DSTF1 is called, IFLAG indicates what is to be done, as follows

=1 The first correction of $\vec{y}_w$ has been made. Compute $\vec{f}_u$ and call DVØA1.

=2 The second correction of $\vec{y}_w$ has been made. Compute $\vec{f}_u$ and call DVØA1.

=3 An extra correction of $\vec{y}_w$ is required. Compute $\vec{f}_v$ and $\vec{f}_w$ and call DSTF1.

=4 Compute partial derivatives, FPS = $\partial\vec{f}_w/\partial\vec{y}_w$, and call DSTF1.

=5 A change in the value of ℓ, see eqs. (1.4)-(1.8), is planned. The user may, if he wishes, override any of these changes, or he can introduce additional changes in ℓ. (In many cases, when a change in ℓ is indicated, the user may want to insure that the value of ℓ is changed for all equations of a certain class.) The procedure for the user to indicate what he wants here has not yet been completely specified. Finally call DSTF1.

=6 IFLAG=6 will only occur if JSTØR=0 (see subsection IV E below), or this type of return is requested through the entry DSTFØ, (see subsection IV C below). The matrix used in the iterative solution of eqs. (1.6)-(1.8) is to be changed when IFLAG=6. (This matrix is always changed after IFLAG=4, but no return with IFLAG=6 is made in this case, since whatever the user wants to do when IFLAG=6, he can do after computing the Jacobian matrix when IFLAG=4.) If JSTØR=0, the user may want to do some sort of set-up whenever this matrix is changed. Reasons for wanting this return when JSTØR≠0 are considered in subsection IV C below.

=7 Possible only when JSTØR=0. The matrix formed when IFLAG=4 or 6 is to be used to get a correction to $\vec{y}_w^{(\ell)}$ given the residuals in eq. (1.8). (More on this in subsection IV E below.) Then call DSTF1.

=8 Fatal error. The value of KEMAX gives the source of the error as indicated in subsection III A. If either DVØA1 or DSTF1 is called, an error message which includes the value of KEMAX is printed, and the program stopped.

=9 It appears that the Jacobian matrix is being computed incorrectly. To continue: CALL DSTF1. (We recommend a careful look at the Jacobian matrix before proceeding.)

KQ is used as in Subsection III A when ℓ=d in eqs. (1.4)-(1.8). When ℓ < d, KQ is less than zero, the value of q for the I-th equation is given by MØD(-KQ(I),100), and the value of d-ℓ for the I-th equation is given by 1-(KQ(I)/100).

KS vector used to indicate whether the I-th equation is implicit, and to indicate the value of ℓ in eqs. (1.4)-(1.8).

KS(I)<0 means the I-th equation is implicit, otherwise the I-th equation is explicit.

|KS(I)|<10 means the initial value of KS(I) is never to be changed.

MOD(|KS(I)|,10) gives the value of d-ℓ for the I-th equation. (Note that one must have d-$\ell \le 4$, and if KS(I) < 0, then ℓ < d. If the initial value of KS(I) does not satisfy these conditions, IFLAG is set =8.) Thus, for example,

KS(I) =0 the I-th equation is never stiff (i.e. ℓ≡d).

=1 on the I-th equation ℓ≡d-1 and is not to be changed.

=10 when starting, the I-th equation is not stiff (ℓ=d), but this can change as the integration proceeds. This is frequently the best value of KS(I) when starting an integration.

=-1 the I-th equation is implicit, ℓ=d-1 and is not to be changed.

The initial value of KS(I) should satisfy $|KS(I)| < 20$. When $|KS(I)| \geq 10$, the value of $|KS(I)|$ is incremented by multiples of 10 in the process of selecting a new value for ℓ.

ASTIF is a logical variable which indicates if any equation is stiff ($\ell < d$)

= .TRUE. $\ell < d$ in some equation
= .FALSE. $\ell = d$ in all equations

The user need not set (and shouldn't change) the value of ASTIF.

JSTØR is an integer used to indicate how the Jacobian matrix is to be stored in FPS.

=0 FPS can be anything, since it isn't used by DSTF. Special returns to the user are made whenever information about the Jacobian would ordinarily be required. See subsection IV E below.

=1 FPS(I,J) contains $\partial f(I)/\partial y(J)$, where $f(I) \equiv f_I$ and $v(J) \equiv y_J$

=2 FPS(I,J) contains $\partial f_x(I)/\partial y_x(J)$, where $f_x(I)$ is the I-th component of f_x and $y_x(J)$ is the J-th component of y_x; see eq. (4.1). JSTØR=2 is frequently best when some of the KS(I)=0, and the Jacobian is computed analytically.

≥3 FPS(I,J) contains $\partial f_w(I)/\partial y_w(J)$; see eq. (4.1). JSTØR ≥ 3 is especially useful when one does not know which equations are going to be stiff and the Jacobian matrix is computed numerically.

<0 indicates that the Jacobian matrix is a band matrix, JSTØR is a vector of dimension 3; where JSTØR(1) indicates the way partials are stored in FPS, JSTØR(2) gives the number of elements to the left of the main diagonal (defined below) and JSTØR(3) gives the number of elements to the right of the main diagonal.

The main diagonal of the Jacobian matrix consists of partials of the form:

$\partial f(I)/\partial y(\hat{I})$, $\hat{I}$ such that $y(\hat{I})\equiv z(I)$, if $|JST\emptyset R(1)|=1$,

$\partial f_x(I)/\partial y_x(\hat{I})$, $\hat{I}$ such that $y_x(\hat{I})\equiv x(I)$, if $|JST\emptyset R(1)|=2$,

$\partial f_w(I)/\partial y_w(\hat{I})$, $\hat{I}$ such that $y_w(\hat{I})\equiv w(I)$, if $|JSTOR(1)|\geq 3$.

In the case of first order equations $\hat{I}\equiv I$. In the description below we use the abbreviated notation, J_L = JST$\emptyset$R(2), J_R = JST$\emptyset$R(3), $J_B = J_L + J_R + 1$ (= the band width), and n, n_x, n_w are the number of components in $\vec{y}$, $\vec{y}_x$, $\vec{y}_w$. One must have JST$\emptyset$R(2) and JST$\emptyset$R(3) $\geq$ 0, and JSTOR(2) + JST$\emptyset$R(3) < n, n_x, or n_w depending on the value of JST$\emptyset$R(1).

JSTOR(1)=-1 FPS(I,J) contains $\partial f(I)/\partial y(K)$, where

$$K = \begin{cases} J & \text{if } \hat{I}\leq J_L \quad J=1,2,\ldots,J_B, \\ J+\hat{I}-J_L-1 & \text{if } J_L<\hat{I}\leq n-J_R \quad J=1,2,\ldots,J_B, \\ J+n-J_B & \text{if } \hat{I}>n-J_R \quad J=1,2,\ldots,J_B, \end{cases}$$

where $\hat{I}$ is defined as above. Equivalently, one can say that $\partial f(I)/\partial y(K)$ is contained in FPS(I,J), where

$$J = \begin{cases} K & \text{if } \hat{I}\leq J_L \quad K=1,2,\ldots,J_B, \\ K-\hat{I}+J_L+1 & \text{if } J_L<\hat{I}\leq n-J_R \quad K=\hat{I}-J_L,\ldots,\hat{I}+J_R, \\ K+J_B-n & \text{if } \hat{I}>n-J_R \quad K=n-J_B+1,\ldots,n. \end{cases}$$

JST$\emptyset$R(1)=-2 Same as JST$\emptyset$R(1)=-1, except f, y, and n are replaced with f_x, y_x, and n_x.

JST$\emptyset$R(1)$\leq$-3 Same as JST$\emptyset$R(1)=-1, except f, y, and n are replaced with f_w, y_w, and n_w.

Note that the above formulas do permit more than J_R elements to the right of the main diagonal when $\hat{I}\leq J_L$ and more than J_L elements to the left of it when $\hat{I}>n-J_R$.

FPS — is a two dimensional array used to store partial derivatives. See usage of JSTØR above, for the way that derivatives are stored in FPS.

IFPS — gives the maximum number of rows in FPS, i.e. FPS is dimensioned FPS(IFPS,?). IFPS must be at least as large as the number of components in $\vec{f}$, $\vec{f}_x$, or $\vec{f}_w$ depending on the value of JSTØR. The number of columns in FPS must be ≥ the number of components in $\vec{y}$, $\vec{y}_x$, or $\vec{y}_w$ when JSTØR = 1, 2, or 3, and ≥ J_B = JSTØR(2)+JSTØR(3)+1 when JSTØR(1) < 0. When JSTØR = 0, FPS and hence IFPS can be anything.

WS — is a vector of dimension IWS (see below) used for working storage. See subsection IV E for information on what is stored in WS.

IWS — is the dimension declared for WS. One must have IWS ≥ n_w+k_j, where $k_j=m_w$ if JSTØR=0, $k_j=m_w*(m_w+3)$ if JSTOR > 0, and $k_j=2*m_w*(J_2+J_3+2)$ if JSTØR(1) < 0; n_w = the largest dimension assumed by $\vec{y}_w$; m_w = the largest dimension assumed by $\vec{f}_w$; J_2 = JSTØR(2), J_3 = JSTØR(3) if KD=1, and with higher order differential equations J_2 and J_3 give the number of equations associated with variables to the left and right of the main diagonal. The precise definitions of J_2 and J_3 are somewhat complicated by the fact that, as in the last sentence in the usage of JSTØR, more than J_3 variables are permitted to the right of the main diagonal when I is close to 1 and more than J_2 are permitted to the left of the main diagonal when I is close to m_w. Thus, J_3 = the maximum value of K (for any I) for which $\partial f_w(I)/\partial w(I+K+\max\{0,J_2-I+1\})$ is contained in FPS and J_2 = the maximum of K for which $\partial f_w(I)/\partial w^{(d-1)}(I-K-\max\{0,J_3-n_w+I\})$ is contained in FPS, where d is the order of the differential equation associated with $w(I-K-\max\{0,J_3-n_w+I\})$.

IW — A vector or working storage with dimension ≥ $2*m_w$, where m_w = the largest dimension assumed by $\vec{f}_w$. IW is not used and thus can be anything if JSTØR=0.

The sample program for use of the stiff integrator is given below.

In the type and dimension information given below, (?) indicate that the dimension depends on how the parameter is used (see just above for parameters which are used only for stiff equations and subsection III A for the other parameters), m ≥ NEQ, and n ≥ total order of the system.

```
INTEGER            NEQ,KD(?),IFLAG,MXSTEP,KSTEP,KEMAX,KQ(m),KS(m),
                   JSTØR(?),IFPS,IWS,IW(?)
LØGICAL            ASTIF
REAL               EP(?),HMINA,EMAX
DØUBLE PRECISIØN   F(m),DT(20,m),FPS(IFPS,?),WS(IWS)
DØUBLE PRECISIØN   T,Y(n),H,DELT,TFINAL,YN(n)
```

A_0 Assign the values for NEQ,KD,MXSTEP,KS,JSTØR,IFPS,IWS, EP,HMINA,DELT, and TFINAL
Assign initial values for T,Y, and H (These will be updated by the integrator. Also, depending on the initial values in KS, KS may be changed.)
Values need not be assigned for IFLAG,KSTEP,KEMAX,KQ, IW,ASTIF,EMAX,F,DT,FPS,WS,YN

A_1 If certain special options are used, they should be set up at this point.

CALL DSTF(NEQ,T,Y,F,KD,EP,IFLAG,H,HMINA,DELT,TFINAL, MXSTEP,KSTEP,KEMAX,EMAX,KQ,YN,DT,KS,ASTIF,JSTØR,FPS, IFPS,WS,IWS,IW)

GØ TØ A_3

A_2 CALL DVØA1

A_3 GØ TØ $(I_1,I_2,\ldots,I_8)$,IFLAG

I_1 CØNTINUE (Special code is required here if certain options are used.)

I_2 Compute the $\vec{f}_v$ and $\vec{f}_w$ parts of F (see eq. (4.1))
IF (.NØT.ASTIF) GØ TØ S_2 (This statement is optional.)

S_0 CALL DSTF1

GØ TØ (S_1,S_2,I_2,S_4,S_5,S_6,S_7,I_8,S_9), IFLAG

S_1 CØNTINUE

S_2 Compute the $\vec{f}_u$ part of F (see eq. (4.1))

GØ TØ A_2

S_4 Compute partial derivatives, FPS (Only $\partial\vec{f}_w/\partial\vec{y}_w$ is required, but one is permitted to compute more; for example, $\partial\vec{f}_x/\partial\vec{y}_x$.)

GØ TØ S_0

S_5 The integrator is planning to change ℓ, see eqs. (1.4)-(1.8). At this point the user can do nothing, or specify other changes in ℓ, or override the changes planned by the integrator. For instructions, see usage for IFLAG=5 above.

GØ TØ S_0

S_6 If JSTØR=0, see subsection IV E below for what to do. If DVØAØ WAS CALLED WITH LMF > 0 (see subsection IV C below), do whatever you wish. If neither of these options is used, then a return with IFLAG=6 should not occur.

S_7 If JSTØR=0, see subsection IV E below for what to do. If JSTØR≠0, then a return with IFLAG=7 should not occur.

S_9 Print an error message indicating that Jacobian is not being computed correctly, and

STØP or GØ TØ S_0

I_3 . . . I_8 Same as in subsection III A.

END

B. Miscellaneous Features Used As in Section III

The relative error test can be used exactly as described in III B.

To restart an integration, everything in III C applies, except that DSTF is substituted for DVØA, and DSTFR is substituted for DVØAR.

The special returns and optional output are available exactly as described in III D.

The direct interpolation works as described in III F. Note that anytime it is permitted to CALL DSTF1 and anytime a return is made from DSTF1, if DAINT1 is called, then T, Y, and F must be saved and then restored if one intends to continue the integration. One change is required when storing information for later retrieval due to the fact that KQ(I) may be < 0. In statement labeled S_1 on page 27 substitute the following for KQ(I): if $\ell \geq$ d-1 on all equations, substitute ABS(KQ(I)); if ℓ may be < d-1, then substitute ABS(MØD(KQ(I)),100)).

One gets more direct control over the integration exactly as described in III G.

The GSTOP feature is inserted into the sample program which uses DSTF, exactly as is described in III H.

The extra output described in III I is available exactly as described there.

Differential equations with side conditions are treated much as described in III J, except that one uses the variational equation entry described in IV D below, and $\vec{y}$ is modified when IFLAG=10 after the call to DSTF1. If this is done, one should compute $\vec{f}_v$ at the same time as $\vec{f}_u$ instead of with $\vec{f}_w$, if it is possible to do so.

C. Optional Output and Special Returns for Stiff Equations

At any place in the program one may insert

CALL DSTFØ(LMF,LØI)

The variables LMF and LØI are both type INTEGER, and both refer to special options as described below. If the user changes one of these parameters, DSTFØ must be called for the change to take effect.

LMF determines if a special return is made with IFLAG=6 whenever the matrix used in the iteration to solve eqs. (1.6)-(1.8) is going to be factored. (This return is not made if the factorization follows immediately after a Jacobian evaluation.) This return enables one to keep track of the number of matrix factorizations, gives one a chance to compute the Jacobian before every factorization (a good idea if the Jacobian is extremely simple to compute) and makes it possible to use the same storage for FPS and WS, see the last paragraph in subsection IV E below.

- >0 Return with IFLAG=6 when the matrix is to be factored without an immediately preceding Jacobian evaluation.
- =0 No change. (The nominal internal flag is set for no return.)
- <0 No return with IFLAG=6 when the matrix is to be factored.

LØI is used to specify that results connected with the iteration to solve eq. (1.6)-(1.8) are to be printed. This information is sometimes useful when debugging a program. The output to be provided is not yet completely specified.

- >0 Give the output. (The size of LØI may determine the amount of output.)
- =0 No change (The nominal internal flag is set for no output.)
- <0 No output.

D. Variational Equations (and Other With Similar Characteristics)

As in subsection III E, it is convenient to consider the case of one additional set of differential equations first. When there are stiff equations it is desirable to have the user label whether this additional set of equations is a system of variational equations as in eq. (3E.2) or is some more general case as permitted by eq. (3E.1). In the former case, the same Jacobian matrix is used for the variational equations as for the original set of equations, while in the latter case the Jacobian matrix $\partial\vec{f}_a/\partial\vec{y}_a$ must be stored, where $\vec{y}_a$ is defined as in eq. (3E.1) and $\vec{f}_a = (f_{m+1},\ldots,f_{m+\mu})^T$. The parameters NVE and EPSV are defined as in subsection III E. In addition, the following parameter is required.

IREF is used to indicate whether another Jacobian is required.

=1 The extra equations are variational equations and thus a new Jacobian is not required. In this case one must have NVE = i*NEQ where i is a positive integer. Also, one should start with KS(I+j*NEQ) = KS(I), I=1,2,...,NEQ, j=1,2,...,NVE/NEQ. This is to insure that the same value of ℓ, see eqs. (1.4)-(1.8), will be used on all equations which utilize the same row of the Jacobian matrix. The integrator will not change the value of ℓ used on a given equation without changing the value of ℓ for all associated variational equations, and vice versa. The user who initiates changes in ℓ, should do the same.

=0 None of the additional equations are permitted to be stiff (i.e. have $\ell < d$).

<0 The extra equations are not variational equations, and -IREF gives the index in FPS where the first element of the Jacobian matrix for the additional equations is stored. Thus, for example, if one is using m rows and n columns of FPS for the Jacobian matrix of the first set of equations and wants to store the Jacobian matrix for the extra set of equations in the next available locations,

then IREF = -(mn+1). In addition to the extra storage required in FPS when IREF < 0, one must also provide extra storage in JSTØR and IFPS, both of which are now vectors, and in WS and IW. Let j_1 be the last location of JSTØR used for the first set of equations. Thus if JSTØR(1) ≥ 0, j_1=1, and if JSTØR(1) < 0, j_1=3. Then the value of JSTØR(j_1+1) indicates the way that information is stored for the extra set of equations in exactly the same way that JSTØR(1) indicates the way information is stored for the first set of equations. Of course, if JSTØR(j_1+1) < 0, then JSTØR(j_1+2) and JSTØR(J_1+3) are used just as JSTØR(2) and JSTØR(3) are used when JSTØR(1) < 0. IFPS(2) gives the maximum number of rows in the Jacobian matrix for the extra equations in exactly the same way that IFPS(1) gives the maximum number of rows for the Jacobian matrix of the first set of equations. Thus the partial derivatives in the first row of the Jacobian for the extra set of equations are stored in FPS(-IREF), FPS(-IREF+IFPS(2)), FPS(-IREF+2*IFPS(2)), ... To indicate the extra storage required in WS and IW, let $\hat{m}_w$ = the largest number of equations in the added set which may be stiff at a given time, and $\hat{n}_w$ = the largest number if dependent variables (from $\vec{y}_a$) which may correspond with a stiff equation. The extra storage required is just what one might expect: for WS, $\hat{n}_w+\hat{k}_j$ where $\hat{k}_j=\hat{m}_w$ if JSTØR(j_1+1) = 0, $\hat{k}_j = \hat{m}_w*(\hat{m}_w+3)$ if JSTØR(j_1+1)>0, and $\hat{k}_j=2*\hat{m}_w*(J_2+J_3+2)$ if JSTØR(j_1+1)<0, where J_2 and J_3 are defined as in the usage of IWS in subsection IV A except with JSTØR(j_1+2) and JSTØR(j_1+3) substituted for JSTØR(2) and JSTØR(3); and for IW, $2*\hat{m}_w$.

Of course, the vectors Y and F are defined as in III E. To use this option the following changes should be made in the sample program given in subsection IV A. (Changes are similar to those given in subsection III E.)

In the dimension information one must have m ≥ NEQ+NVE, and n ≥ the total order of the system including the equations which have been added. With mixed orders KD must have dimension ≥ m, otherwise KD can be a scalar. One should add NVE and IREF to the variables specified to be of type INTEGER, and EPSV to those specified to be of type REAL.

At A_0 initial values for NVE, EPSV, and IREF must be assigned
At A_1 (before the call to DSTF)

```
CALL DSTFV(NVE,EPSV,IREF)
```

Code from A_3 to the GØ TØ A_2 just below S_2 should be replaced with

A_3	GØ TØ (I_1,I_2,...,I_{10}), IFLAG
I_1	CØNTINUE (Special code is required here if certain options are used.)
I_{10}	Compute the $\vec{f}_v$ and $\vec{f}_w$ parts of F(I), for 1 ≤ I ≤ NEQ
S_0	CALL DSTF1 GØ TØ (S_1,S_2,I_{10},S_4,S_5,S_6,S_7,I_8,S_9,S_{10},S_{11}), IFLAG
S_1	CØNTINUE Compute the $\vec{f}_u$ part of F(I), for 1 ≤ I ≤ NEQ GØ TØ A_2
S_{10}	Compute subexpressions which depend only on $\vec{y}$, see eq. (3E.1) ($\alpha(t,y)$ and $\vec{b}(t,\vec{y})$ in the case of variational equations)
I_2	Compute $\vec{f}_v$ and $\vec{f}_w$ parts of F(I), I > NEQ GØ TØ S_0
S_{11}	CØNTINUE
S_2	Compute $\vec{f}_u$ part of F(I), I > NEQ If (IFLAG-2) any, S_1, A_2

Note that if F(I) depends only on $\vec{y}$ (thus not on $\vec{y}_a$) for I > NEQ, then none of the equations should be treated as stiff (so have KS(I)≡0 for I > NEQ and IREF=0), and the calculation of the $\vec{f}_u$ part of F(I) at S_2 can be skipped when IFLAG=2.

If one is not using the end of step return (LSS) in subsection D, simply set $I_9 \equiv A_2$.

The more general case in eq. (3E.4) requires that IREF be a vector, and that NVE and EPSV be defined as they are just below eq. (3E.4). The K-th additional set of equations is treated as indicated by IREF(K).

IREF(K)=j>0 indicates that the K-th extra set of equations, is a set of variational equations associated with the j-th set of equations. (The (j+1)-st set of equations ≡ j-th extra set of equations.) In this case one must have NVE(K+2)=i*μ_{j-1}, where i is a positive integer and μ_{j-1}=NVE(j+1) if j>1 and =NEQ if j=1. As before, the initial values of KS(I) should be the same for all equations which use the same row of a given Jacobian matrix.

IREF(K)=0 indicates that there are no stiff equations in the K-th extra set of equations.

IREF(K)<0 indicates that the K-th extra set of equations is not variational equations, and specifies the starting point in FPS where the Jacobian matrix is to be found as described under IREF<0 above. As before, the way the Jacobian is stored in FPS is indicated in JSTØR starting with JSTØR(j_K+1), where j_K is the last value of JSTØR used by the K-th set of equations. (IREF(K)≥0 does not use any storage in JSTØR.) Also, IFPS(K+1) gives the maximum row dimension of the Jacobian stored in FPS. The extra storage required in WS and IW is computed using the obvious generalization of the formulas given earlier under IREF<0.

IREF must satisfy the following: If IREF(K)<0 then IREF(i)≤0 for all i<K; if IREF(K)=j>0, then j>IREF(i) for all i<K and either j=1, or IREF(j-1)<0.

In order to handle several additional sets of equations, one should modify the changes given earlier in this subsection by dimensioning IREF, and then changing the code from I_1 on, as follows.

I_1 CØNTINUE (Special code is required here if certain options are used.)
K=1
GØ TØ I_2

I_{10} IF (K.EQ.1) GØ TØ I_2

E_1 IF (IREF(K-1).EQ.0) GØ TØ S_0

I_2 Compute $\vec{f}_v$ and $\vec{f}_w$ parts for the K-th set of F(I)'s

S_0 CALL DSTF1
GØ TØ (S_1,S_2,E_1,S_4,S_5,S_6,S_7,I_8,S_9,S_{10},S_{11}), IFLAG

S_1 CØNTINUE
GØ TØ S_{11}

S_{10} K=NVE(2)+1
Compute subexpressions for the K-th set of equations (NVE(2)-th additional set) which depend only on components of Y from previous sets.
GØ TØ E_1

S_{11} Compute $\vec{f}_u$ part for the K-th set of F(I)'s.
GØ TØ A_2

S_2 Compute $\vec{f}_u$ part of all F(I)'s (Those components of F which depend only on components of Y from an earlier set of equations need not be computed at this time.)
GØ TØ A_2

E. Jacobian Has Special Structure (e.g. Sparse)

By setting JSTØR=0, the user can take control over the solution of eqs. (1.6)-(1.8). In this subsection, the following notation is used, where $\vec{y}_w$, $\vec{z}_w$, and $\vec{f}_w$ are defined at the beginning of Section IV.

$$\vec{\eta} = (\eta_1,\eta_2,\ldots,\eta_n)^T = \vec{y}_w,$$

$$\vec{\zeta} = (\zeta_1,\zeta_2,\ldots,\zeta_m)^T = \vec{z}_w,$$

$$\vec{\varphi} = (\varphi_1,\varphi_2,\ldots,\varphi_m)^T = \vec{f}_w,$$

ℓ_i, i=1,2,...,m is the value of ℓ (see eqs. (1.4)-(1.8)) used with ζ_i,

$$\vec{\xi} = (\xi_1,\xi_2,\ldots,\xi_m)^T = (\zeta_1^{(\ell_1)},\ldots,\zeta_m^{(\ell_m)})^T,$$

d_i is the order of the differential equation corresponding to φ_i,

q_i is the integration order used on the equation corresponding to φ_i,

$$p_{ij} = \partial\varphi_i/\partial\eta_j \quad i=1,2,\ldots,m,\ j=1,2,\ldots,n.$$

(4E.1)

The option described here presumably is of interest only if there is something special about the matrix (p_{ij}) which the user wants to take advantage of in obtaining the solution of eqs. (1.6)-(1.8). An example of such a situation is the case when most of the p_{ij} are zero. We restrict our attention here to the case when the user linearizes $\vec{\varphi}$, and solves the resulting linear problem as an approximation to the solution of eqs. (1.6)-(1.8). There are other ways the user might approach the problem of getting an approximate solution to eqs. (1.6)-(1.8).

Consider first the case when $d_i \equiv 1$, and thus n=m, and $\ell_i \equiv 0$. The user solves for a correction, $\delta\vec{\xi}$, to $\vec{\xi}$ using the formula

$$A\delta\vec{\xi} = \vec{r} = (r_1,r_2,\ldots,r_m)^T \qquad (4E.2)$$

where $A=(a_{ij})$, $a_{ij}=p_{ij}$ if $i\neq j$ and $a_{ii}=p_{ii}-WS(i)$, $WS(i)=(1/h)\beta_{q_i,1}$ if the i-th equation is explicit and =0 if the i-th equation is implicit, $r_i=WS(n+i)$= the residual that results from subtracting the left side of eq. (1.8) from the right side, and the contents of WS are computed by DSTF. When IFLAG=6 and after computing the Jacobian when IFLAG=4, the user should compute the matrix A, and do any preliminary set up that will be used for later iterations. When IFLAG=7, the user should solve eq. (4E.2), and store the vector $\vec{\delta\xi}$ in the storage previously occupied by $\vec{r}$. In all of these cases, one continues by calling DSTF1.

When some of the d_i are greater than one, $n=d_1+d_2+\cdots+d_m > m$, and an nxn linear system results from the linearizing eqs. (1.6)-(1.8). However, the special structure of eqs. (1.6) and (1.7) makes it possible to eliminate n-m variables in eq. (1.8). One is then left with the mxm system (4E.2) to solve, and DSTF solves for the remaining varaibles. Usage is the same as described above, except that the matrix A is more complicated to obtain, and the contents of WS require a more general definition. Let $k_0=0$, $k_i=d_1+d_2+\cdots+d_i$, $i>0$. WS(j) contains a coefficient used in obtaining the value of η_j. More specifically, $WS(k_{i-1}+1)$ to $WS(k_i)$ contain coefficients used for $\zeta_i,\zeta_i',\ldots,\zeta_i^{(d_i-1)}$. The first ℓ_i cells starting with $WS(k_{i-1}+1)$ (if $\ell=0$, then no cells) contain

$$h^{\ell_i}\gamma_{q_i,\ell_i},\ldots,h\gamma_{q_i,1},$$

the next cell contains

$$h^{\ell_i-d_i}\beta_{q_i,d_i-\ell_i}$$ (if the i-th equation is explicit),

or 0 (if implicit),

the remaining cells (none if $\ell_i=d_i-1$) contain

$$h^{-1}\beta_{q_i,1},\ldots,h^{-(d_i-\ell_i-1)}\beta_{q_i,d_i-\ell_i-1}.$$

As before WS(n+i) contains the i-th residual, r_i, to eq. (1.8). From the structure of eqs. (1.6)-(1.8) and the fact that the initial estimates are such that the residuals for eqs. (1.6) and (1.7) are zero, it follows that A can be defined as follows. ($p(i,j)\equiv p_{ij}$)

$$a_{ij}=\left[WS(k_{j-1}+1)*p(i,k_{j-1}+1)+\cdots+WS(k_{j-1}+\ell_j)*p(i,k_{j-1}+\ell_j)\right]$$

$$+\begin{cases} p(i,k_{j-1}+\ell_j+1) & \text{if } i\neq j \\ p(i,k_{j-1}+\ell_j+1) - WS(k_{j-1}+\ell_j+1) & \text{if } i=j \end{cases}$$

$$+\left[WS(k_{j-1}+\ell_j+2)*p(i,k_{j-1}+\ell_j+2)+\cdots+WS(k_j)*p(i,k_j)\right]$$

where either of the sums in brackets may not contain any terms (and hence be equal to zero) depending on the values of ℓ_j and d_j. (Note that $k_j=k_{j-1}+d_j$; and $p(i,k_{j-1}+1),\ldots,p(i,k_j)$ are used in forming a_{ij}.)

When JSTØR≠0, DSTF selects the value of ℓ_i automatically, but when JSTØR=0 the value of ℓ_i is only changed if the user initiates the change.

The first n_w and the next m_w locations in WS are always used as described above. When JSTØR≠0, the next $2*m_w$ locations are used as working storage by the linear algebra subroutines. And

finally, the next m_w^2 locations (if JSTØR>0) or the next $m_w*(2*\alpha_B-1)$ locations (if JSTØR(1)<0, and where α_B denotes the band width in the matrix A) are used to store the matrix A and the factored form of A. When there are variational equations this same pattern is followed for the additional equations, always starting with the first unused location in WS.

If there are no variational equations, one always recomputes FPS whenever A is factored (use LMF>0 in the options described in subsection IV C and compute FPS when IFLAG=6), and FPS is not used for anything but forming the matrix A in eq. (4E.2), then one can use part of the storage required for WS to hold FPS. Use an EQUIVALENCE statement to use the same location for FPS(1,1) and WS(k), where k≥j+1 and j = maximum value ever attained by n_w+3*m_w.

F. Control of the Type of Past Information Stored

Since we do not as yet have a good automatic algorithm for changing ℓ when ℓ=d (see eqs. (1.4)-(1.8)), and even if we did there would still be cases when the user might do a better job, a feature is included which enables the user to change ℓ at anytime in the computation. To use this feature one executes the following statement at anytime

```
CALL DSTFC
```

and then continues as one would if this statement were absent. At the first opportunity DSTF will return control to the user with IFLAG=5. The user can then change the vector KS as described in subsection IV A under the usage of IFLAG=5.

G. User Control of Jacobian Evaluations

If the optional return with IFLAG=6 does not let the user evaluate the Jacobian matrix as often as he wants to, he can evaluate the Jacobian more frequently using the feature described here. At anytime execute

```
CALL DSTFJ
```

and then continue as usual. At the first opportunity DSTF will return control to the user with IFLAG=4 at which time the user should compute the Jacobian matrix and call DSTF1. Note that it is permitted to change FPS without the new Jacobian being used in the iteration to solve eqs. (1.6)-(1.8). (We assume that the same storage is not being used for FPS and WS.) For example, one may want to do this when integrating variational equations. Of course, if this is the case one would not call DSTFJ.

H. Numerical Evaluation of the Jacobian Matrix

We plan to provide a subroutine DPART which the user can call anytime he needs to compute partial derivatives for the Jacobian matrix. More work is required before specifications for this subroutine can be given.

V. CONCLUDING REMARKS

We plan to use parts of this document in the subroutine write-up to be prepared later. However, when this work is completed it is likely that the usage will be slightly different, and the write-up of the subroutines will include additional material to aid people who use them. In addition, a shorter write-up will be available for those with simple problems. We also plan to give some thought to providing a program which will only require the user to provide the code for the derivative calculations, and which will enable the user to specify things such as form of output, error tolerance, initial conditions, etc. through special input parameters. Of course, such a program would not have near the flexibility of the software described here.

It would have been nice to include the ability to automatically select the type of error test to be used. That is, should the local error be kept less than some specified ϵ or less than ϵ times some function, and if so, what function? The answer, of course, depends on the global effect of local errors, and these effects seem impossible to estimate automatically without an inordinate amount of computation. In [1], we give suggestions on how to select the type of error test. Unfortunately, users avoid reading subroutine write-ups if they can, and they frequently can. The result is that users frequently use a type of test that results in an inefficient computation. Is there a good solution to this problem?

For completeness, we mention one more feature which will be added if there is sufficient interest. This feature will permit the user to group his differential equations, with all equations in a group integrated with the same stepsize, but with the possibility that different stepsizes will be used for equations in different groups. All stepsizes would be some power of two times the smallest stepsize. This feature would introduce an extra version of DVØA (and DSTF) if it requires a significant amount of extra code in DVØA, or if it has any more than negligible effect on the integration overhead when the feature is not used. All of the features including already have satisfied these two constraints.

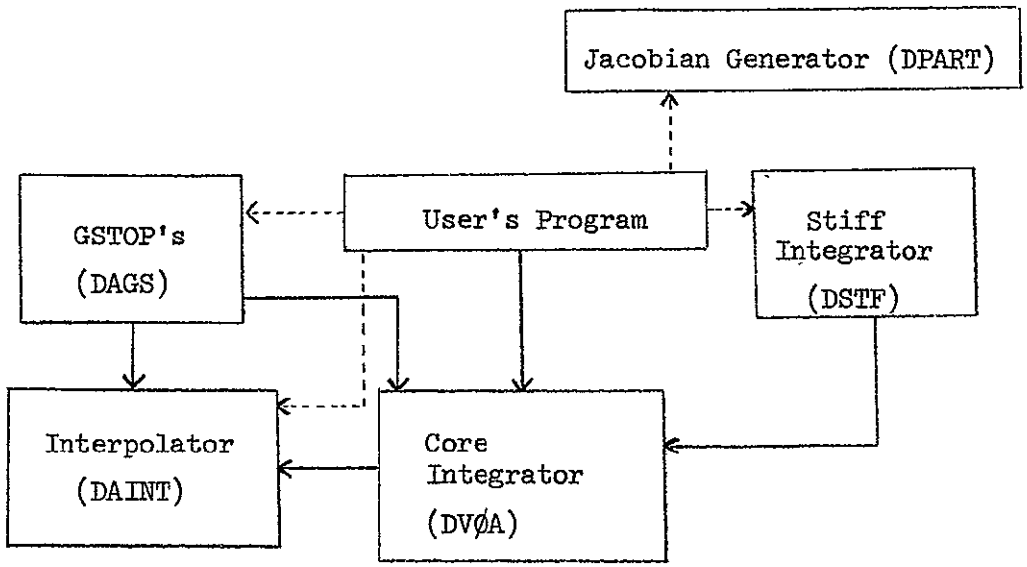
ACKNOWLEDGEMENT

Thanks are due the following people who have either used experimental versions of our integrators (and in the process taught us things that could not be learned from simple test cases), or have discovered things that were awkward or impossible to do that we would never have thought of without their experience: Peter Breckheimer, John Light, Melba Nead, Carl Sauer, and Bill Sinclair. The design of the stiff integrator has benefited from the excellent work that C. W. Gear has done in this area.

REFERENCES

1. Krogh, Fred T. VODQ/SVDQ/DVDQ - Variable Order Integrators for the Numerical Solution of Ordinary Differential Equations. Section 314 subroutine write-up, Jet Propulsion Laboratory, Pasadena, Calif., May 1969.

2. Krogh, Fred T. On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations. Section 314 Technical Memorandum No. 217 (Revised), Jet Propulsion Laboratory, Pasadena, Calif., October 1970.



Dashed lines are for optional calls.

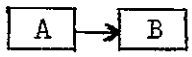
 means A calls B, and B returns control to A.

FIGURE 1:

Communication Between the Subroutines and the User

NASA — JPL — Coml., L.A., Calif.