Eric Grosse
17 Jun 1978

Computing Science Technical Report #33

A User's Guide to DODES, a Double Precision Ordinary Differential Equation Solver

N. L. Schryer

August 1975

A User's Guide to DODES, a Double Precision Ordinary Differential Equation Solver.

N. L. Schryer

Bell Laboratories,
Murray Hill, New Jersey 07974

ABSTRACT

DODES (Double precision Ordinary Differential Equation Solver) is a package of portable FORTRAN subprograms for integrating first order initial value problems of the form

$$\frac{dx}{dt} = f(t,x), \qquad x(t_s) = x_s \tag{1}$$

where $x(t)$ is a vector valued function of time t, f is a vector valued function of t and x, and x_s is a vector of initial conditions. These subprograms allow easy user control over both the accuracy and the output of the integration process.

The algorithm used is a variable order, variable step-size extrapolation scheme augmented by several mechanisms for dealing with discontinuities in the derivatives of the solution. Previous extrapolation based differential equation solvers lack one or more of these features of DODES. Thus, DODES is a more robust, efficient and reliable method for solving (1).

A User's Guide to DODES, a Double Precision Ordinary Differential Equation Solver.

N. L. Schryer

Bell Laboratories,
Murray Hill, New Jersey 07974

1. Introduction

DODES (Double precision Ordinary Differential Equation Solver) is a package of portable FORTRAN subprograms for integrating first order initial value problems of the form

$$\frac{dx}{dt} = f(t,x) \tag{1.1}$$

subject to initial conditions

$$x(t_s) = x_s \tag{1.2}$$

where $x(t)$ is a vector valued function of time t, f is a vector valued function of t and x, and x_s is a vector of initial conditions. These subprograms allow easy user control over both the accuracy and the output of the integration process. The algorithm used is a variable order, variable step-size extrapolation scheme which is locally "optimal". That is, at each step in the integration procedure, the order and step-size are chosen to minimize the cost per unit time-step. This extrapolation scheme is augmented by several mechanisms for dealing with discontinuities in the derivatives of the solution. This makes DODES a robust, efficient and reliable method for solving (1.1) subject to (1.2).

The basic algorithm used by DODES is extrapolation to the limit of Gragg's modified mid-point rule [8]. This technique has been used previously in several differential equation solvers, [2, 3, 17]. It has been found [5, 11, 19] to be efficient and competitive with other methods [7, 12] for solving (1.1).

The step-size and order monitor described in [14] is used to drive the integration process. This makes the overall procedure far more robust and reliable, as well as increasing its efficiency. The efficiency (cost) of DODES is substantially independent of the initial time-step chosen by the user. It is sufficient to have the initial time-step merely within a few orders of magnitude of the "correct" value.

The techniques of [2] and [3] are fixed (14^{th}) order, variable step-size methods, with rather crude step-size monitors. The fixed-order step-size monitor of [17] was a great improvement over those used in [2] and [3]. However, one assumption made by [17] is often violated in practice. This situation typically arises when the solution of (1.1) is pleasantly smooth in some regions, while being exceedingly "kinky" in others. In such cases, the use of [17] results in rather inefficient integration. DODES has a locally optimal monitor for both order and step-size, and a mechanism for dealing with "kinky" solutions. This makes DODES more efficient, and more reliable, than [2], [3] or [17] - in some cases by more than an order of magnitude. A thorough comparison of the performance of DODES with that of [2], [3], [7], [12] and [17] is made in [15].

The program unit DODES has been successfully verified for adherence to a portable subset of ANS Standard FORTRAN, called PFORT, using the PFORT verifier [13]. This virtually guarantees that the program will successfully compile on any existing ANS conforming FORTRAN compiler. DODES uses the function subprogram I1MACH [4] from the PORT library [6] to obtain all machine dependent parameters. Thus, DODES may easily be ported to any machine which supports ANS FORTRAN. DODES also uses the storage allocator [9] and the error handling facility [10] of the PORT library.

Section 2 briefly discusses the use of extrapolation and Gragg's modified mid-point rule to solve (1.1), as well as the application of the monitor [14] to drive the process. Section 3 presents the calling sequences, and a description of the arguments, for the subroutines DODES, DODES1 and DODES2. These three routines provide various levels of detailed control over the integration process. Section 4 gives several examples of the use of DODES. Finally, section 5 discusses various implementation details which some users of DODES may find useful.

2. The Algorithm

The underlying discretization process used is Gragg's modified mid-point rule [8]. When asked to integrate (1.1) between time t_l and time t_r, using $2N$ time-steps, that rule sets $h=(t_r-t_l)/N$, $x_0=x(t_l)$, and $x_1=x(t_l)+\frac{h}{2}f(t_l,x_l)$. Then, for $i=2,\cdots,2N$, the quantities

$$x_i = x_{i-2} + hf(t_l + (i-1)\frac{h}{2}\ ,\ x_{i-1})$$

are computed. This gives x_{2N} as an approximation to $x(t_r)$. In fact, we have [16]

$$T(h) \equiv x_{2N} = x(t_r) + \sum_{j=1}^{\infty} \tau_j h^{2j} \tag{2.1}$$

where the τ_j are unknown vectors which are independent of h. Thus, for small h, each component of $T(h)$ resembles a polynomial in h^2. We want to obtain $T(0)=x(t_r)$ accurately.

The process of extrapolation is easily described. Let a sequence of h's be defined by $h_i=h_0/N_i$, $i=1,2,3,\cdots$ where $h_0=t_r-t_l$ and the N_i form a monotone increasing sequence of positive integers. Bulirsch and Stoer show in [1] that given an operator $T(h)$ satisfying (2.1), and such a sequence h_i, the value at $h=0$ of the polynomial of degree m which interpolates $T(h_i)$ for i=0, ... , m, is given by T_m^0 which is determined from the following recursion relations

$$T_0^i = T(h_i) \quad \text{for} \quad 0 \leqslant i \leqslant m$$

and

$$T_j^i = T_{j-1}^{i+1} + (T_{j-1}^{i+1} - T_{j-1}^{i})/\{(h_i/h_{i+j})^2 - 1\} \tag{2.2}$$

for $0 \leqslant i \leqslant m-j$, $1 \leqslant j \leqslant m$.

If the T_j^i are organized into a *lozenge* of the form

$$\begin{array}{lccccccc}
T(h_0)=T_0^0 & & & & & & & \\
T(h_1)=T_0^1 & T_1^0 & & T_2^0 & & & & \\
T(h_2)=T_0^2 & T_1^1 & & T_2^1 & & T_3^0 & & T_4^0 \\
T(h_3)=T_0^3 & T_1^2 & & T_2^2 & & T_3^1 & & T_4^1 \quad T_5^0 \\
T(h_4)=T_0^4 & T_1^3 & & T_2^3 & & T_3^2 & & \\
T(h_5)=T_0^5 & T_1^4 & & & & & &
\end{array}$$

then the above recursion relation (2.2) expresses each element of the j-th column (j>0) in

terms of its two neighbors in column j-1. In [1] it is also shown that the error obeys

$$|T_j^i - T(0)| \leq M_{j+1}\,(h_i \cdots h_{i+j})^2 \tag{2.3}$$

for some constants M_{j+1}. Finally, it is shown that for sufficiently small h_0,

$$|T_j^i - T(0)| \approx \left[1 + \frac{1}{(h_i/h_{i+j+1})^2 - 1}\right] |T_j^{i+1} - T_j^i| \tag{2.4}$$

and thus we can estimate the error in T_j^i. A similar result is established for interpolation by rational functions [1]. From (2.3), we define the *order* in column j to be 2(j+1). The *level* of extrapolation in a lozenge is defined to be the number of entries in the first column of that lozenge. The value $h_0 = t_r - t_l$ is referred to as the *time-step* while the h_i are called *sub-steps*. Extrapolation approximates the $x(t_r)$ values accurately, but does not accurately approximate $x(t_l + nh_i)$ for $0 < n < N_i$.

Extrapolation of Gragg's modified mid-point rule gives a process of arbitrarily high order. Also, not only does the extrapolation procedure generate accurate (high-order) results, it also provides reliable error estimates, via (2.4) above, for those results.

Relations (2.3) and (2.4) describe both the rate of convergence in each column of the extrapolation lozenge and the actual error in each element of the lozenge. This is sufficient information to determine which columns of the lozenge are locally "optimal" and what step-size h_0 is locally "optimal." The step-size and order monitor [14] makes use of these facts to provide an "optimal" choice of step-size and order for the extrapolation process. Since the extrapolation process assumes that the solution x(t) has many continuous derivatives, the monitor of [14] also has several mechanisms which allow efficient integration even through singularities.

The monitor of [14] is the main driver of the solution process. Basically, all the package DODES does is provide the monitor with subroutines for computing approximate solutions of (1.1) using Gragg's modified mid-point rule, for computing error tolerances, and for handling the step-wise output of the integration process.

3. The Subprograms

This section describes the subroutines DODES, DODES1 and DODES2. These subroutines provide various levels of control over the integration process. A function subprogram DODESE is described for controlling the accuracy of the computed solution.

The simplest way to solve (1.1) is to use the

SUBROUTINE DODES(F, X, NX, TSTART, TSTOP, DT, ERRPAR, HANDLE)

The input to this subroutine consists of:

F - A subroutine for computing the right-hand side of (1.1). CALL F(T,X,NX,FTX) should return FTX(I) = the I-th component of the vector f(T,X), for I=1,...,NX. The subroutine F should be declared EXTERNAL in the program calling DODES.

X - The initial values for the solution, X=x(TSTART).

NX - The length of the solution vector X.

TSTART - The initial time, that is, X=x(TSTART).

TSTOP - The final time, the point in time at which integration should stop.

DT - The initial time-step to be used. The performance of DODES is quite independent of the value of DT chosen by the user. It is sufficient to merely have DT within a few orders of magnitude of being "correct". The value of DT will be automatically adjusted by DODES, during the integration process, to achieve the accuracy desired at the least possible cost.

ERRPAR - A REAL vector of length 2 for use in controlling the accuracy of the computed solution. Specifically, each component X(I) of the solution will be computed to within an absolute error of

ERRPAR(1) * DABS(X(I)) + ERRPAR(2)

for I=1,...,NX, at each time-step. This error request must always be positive.

HANDLE - A subroutine for interacting with the integration process. At the end of each time-step there is a good deal of information which DODES has internally available. This information includes such items as the solution x at a new point in time, an error estimate for that computed solution, and an "optimal" DT for the next time-step. The subroutine HANDLE is used by DODES to communicate this information to the user. On the other hand, the user may also communicate with DODES through HANDLE, as described below. Thus, HANDLE gives the user a "handle" on both the results of the integration and on the way DODES does its job. At the end of each time-step, DODES will execute the statement

CALL HANDLE(T0, X0, T1, X1, NX, DT, TSTOP, E)

where X0=x(T0) is the value of the solution at the end of the previous time-step and X1=x(T1) is the value of the solution at the end of the current time-step. HANDLE should be declared EXTERNAL in the program calling DODES. If T0=T1 then DODES failed to converge using the previous value of DT, the values in X1 are garbage, and integration from time T0 will be re-tried with the current value of DT. If the solution being obtained is reasonably smooth, once the integration process has made progress (T0 ≠ TSTART), such restarts should not occur. The other input to this subroutine follows:

NX - The length of the solution vector X, same as in the call to DODES.

DT - The proposed "optimal" size for the next time-step.

TSTOP - The current value of the final time for the integration.

E - A REAL array of estimates for the errors in the values of X1 for the single current time-step. The error in X1(I) is E(I) for I=1,...,NX, assuming that X0 is exact at the beginning of the time-step.

On return from HANDLE, if T1=TSTOP, integration will cease and DODES will return control to its caller. The user may alter any of the values X1, DT or TSTOP before returning from HANDLE. HANDLE can do many things - print the solution out, save it for later processing, simply return, create plots of the solution or whatever the user desires.

The output from DODES consists of

X - The final value for the solution, X=x(TSTOP).

TSTOP - May be altered by the user supplied subroutine HANDLE.

DT - Proposed "optimal" size for the next time-step, if any.

The subroutine DODES has 7 error states:

1 Must have NX $\geq$ 1.

2 Must have Sign(DT) = Sign(TSTOP - TSTART) for the input values of those variables.

3 Must have TSTART+DT $\neq$ TSTART, in floating-point arithmetic, for the input values of those variables.

4 HANDLE cannot alter DT and/or TSTOP so as to violate the condition Sign(DT) = Sign(TSTOP - T1).

5 HANDLE cannot alter DT so that T1+DT=T1 in floating-point arithmetic. This error is recoverable.

6 The error tolerance for each component of the solution vector X must be positive. This error is recoverable.

7 The step-size monitor has chosen a value for DT which obeys T1+DT=T1 in floating-point arithmetic. This may simply mean that the system (1.1) is "stiff". In that case, a "stiff" differential equation solver should be used, see [7] or [18]. Another possible explanation is that the subroutine F for computing the right-hand side of (1.1) is improperly coded - the "function" it actually computes is not really a decent function.

The amount of scratch space allocated by DODES is

$$S(DODES) \leq 165 + 26*NX + Max(3*NX + S(F), 10*NX + 30, S(HANDLE)) \quad (3.1)$$

INTEGER words, where S(SUBPROG) is defined to be the number of INTEGER words of scratch storage allocated by subprogram SUBPROG.

More detailed error control over the integration process is obtained by using the

SUBROUTINE DODES1(F, X, NX, TSTART, TSTOP, DT, ERROR, ERRPAR, HANDLE, GLBMAX, ERPUTS)

The extra arguments (ERROR, GLBMAX and ERPUTS) in this subroutine provide direct user control over the accuracy of the integration process.

There are several possible options available for error specification. First, the user, via the subprogram ERROR, may specify literally any accuracy requirement he wishes for the solution. Second, there are several popular methods of error control which are controlled by the switches GLBMAX and ERPUTS, and implemented by the subprogram DODESE.

The error control provided in the subroutine DODES is based on the local value of the variables. That is, the error acceptable in X(I) is

$$ERRPAR(1)*DABS(X(I)) + ERRPAR(2) \quad (3.2)$$

which depends only upon the current value of X(I). However, in some cases it is desirable to have the error in a component of the solution depend upon the maximum absolute value that component has attained since the start of the integration process at time t=TSTART. In that case, the error desired in X(I) is of the same form as (3.2) above, but with X(I) replaced by

$$\max_{t \in [TSTART,\ T]} |x_I(t)| \quad (3.3)$$

That option is controlled by the switch GLBMAX, as described below.

The error control provided in the subroutine DODES is an *error per time-step* criterion. This can be rather bad if the time-steps taken during the solution process get very small - many time-steps will be taken and the errors may pile up in unacceptable amounts. Another error option is to use an *error per unit-time-step* criterion. By making the error tolerance in X(I)

look like

$$DABS(DT) * (ERRPAR(1) * DABS(X(I)) + ERRPAR(2)), \qquad (3.4)$$

when the time-step gets small, so does the error requirement. However, when using the error per unit-time-step criterion, the reverse argument holds - when DT is large, so is the error tolerance. The error per time-step versus unit-time-step option is controlled by the switch ERPUTS as described below.

The inputs for DODES1 are as previously described in DODES, with the following additions:

ERROR - This is a subprogram for computing accuracy requirements for the solution process. It must be declared EXTERNAL in the subprogram calling DODES1. DODES uses ERROR in determining when the extrapolation process has "converged". When DODES has computed a tentative solution vector, it also has an estimate available for the error in that vector. DODES uses the function subprogram ERROR to ask the user if the computed solution vector is sufficiently accurate. ERROR must have the form

LOGICAL FUNCTION ERROR(X, NX, T, DT, ERRPAR, ERPUTS, E)

where X=x(T) is the solution vector of length NX for which an error tolerance is to be supplied by ERROR. The other inputs to ERROR are

DT - The time-step used to obtain X=x(T).

ERRPAR - A REAL vector of length two, as passed to DODES1, or as modified by a previous call to ERROR. (See below.)

ERPUTS - This LOGICAL variable has the same value as the input variable ERPUTS passed to DODES1. (See below.)

E - This REAL vector gives the absolute accuracy of the solution X, as computed by DODES1. The absolute error in X(I) is E(I), for I=1,...,NX, for the single current time-step.

The value returned by ERROR to DODES1 is

- .TRUE. if the tentative solution X is satisfactory to the user; otherwise .FALSE. .

The output from ERROR is

ERRPAR - This vector may be altered, if desired.

E - The vector of REAL absolute errors the user will tolerate in the proposed solution vector X. E(I) is the acceptable absolute error in X(I), for I=1,...,NX. All of the returned E(I) must be positive.

ERRPAR - This REAL vector of length two will be passed to ERROR, just as received by DODES1. Possible uses for this vector are shown in (3.2) and (3.4).

GLBMAX - If GLBMAX is .TRUE., then the global maximum absolute value (3.3) of each component of the solution X is to be recorded as described in Section 5. If the ERROR subprogram supplied by the user is DODESE, then either the global maximum absolute value or the local value of the solution will be used in the error tolerance, depending on whether GLBMAX is .TRUE. or .FALSE. . If the user supplies his own ERROR subprogram, then the error tolerance is at his discretion.

ERPUTS - If the ERROR subprogram supplied by the user is DODESE, then either an error per unit-time-step criterion like (3.4), or an error per time-step criterion like (3.2), will be used, depending on whether ERPUTS is .TRUE. or .FALSE. . If the user supplies his own ERROR subprogram, then the error tolerance for each component

of the solution should be proportional to DABS(DT) if ERPUTS=.TRUE., and should not have this property otherwise.

The output from DODES1 is the same as that for DODES with the additional possibility that the user may alter ERRPAR through his subprogram ERROR.

The error states for DODES1 are the same as those for DODES. The scratch storage allocated by DODES1 is

$$S(\text{DODES1}) \leq 165 + 2*NX*(13 + (\text{If (GLBMAX) then 1, Else 0})) + \text{Max}(3*NX + S(F), 10*NX + \text{Max}(30, S(\text{ERROR})), S(\text{HANDLE})) \quad (3.5)$$

INTEGER words. Note that (3.1) is a special case of (3.5), with GLBMAX=.FALSE. , and S(ERROR)=S(DODESE)=0.

Finally, control over the order of the integration procedure is allowed by

SUBROUTINE DODES2(F, X, NX, TSTART, TSTOP, DT, ERROR, ERRPAR, HANDLE, GLBMAX, ERPUTS, KMAX, MMAX)

The additional arguments in this subroutine (KMAX and MMAX) control the maximum order (2*KMAX) and the maximum level of extrapolation (MMAX) used by the process. The extrapolation lozenge is computed in such a manner that only its lower edge need be stored. Thus, if the length of the lower edge is truncated (limited), the level of extrapolation (the number of entries computed in the first column of the lozenge) may get arbitrarily large without increasing the amount of memory needed to store this truncated lozenge. The subroutine DODES2 allows user control over both the maximum number of columns retained in the lozenge, and the maximum level of extrapolation permitted. The arguments of DODES2 are the same as those of DODES1 with the following additions:

KMAX - The maximum number of columns allowed in the extrapolation lozenge. The maximal order that DODES2 can achieve is then 2*KMAX.

MMAX - The maximum number of levels of extrapolation permitted. MMAX $\geq$ KMAX + 2 is required and MMAX $\geq$ KMAX +4 is a good idea.

There are two additional error states for DODES2:

9 Must have KMAX $\geq$ 1.

10 Must have MMAX $\geq$ KMAX + 2.

Scratch space of length

$$S(\text{DODES2}) \leq 5*KMAX + 7*MMAX + 3 + 2*NX*(KMAX + 3 + (\text{If (GLBMAX) then 1, Else 0})) + \text{Max}(3*NX + S(F), NX*KMAX + \text{Max}(3*KMAX, S(\text{ERROR})), S(\text{HANDLE})) \quad (3.6)$$

INTEGER words is allocated by DODES2. Note that (3.5) is a special case of (3.6), with KMAX=10 and MMAX=16.

The subroutines DODES1 and DODES2 require a function subprogram to specify the error tolerable in the solution. The subroutine DODES uses the default function subprogram DODESE. This routine is listed below to provide an example of what a user supplied ERROR subprogram might look like. Notice that DODESE can provide error control on a per time-step or per unit-time-step basis. DODESE can also use either the local value or the global maximum absolute value of the solution in determining the error control.

```
      LOGICAL FUNCTION DODESE(X,NX,T,DT,ERRPAR,ERPUTS,E)
C
C  STANDARD ERROR ROUTINE FOR DODES WITH THE OPTION FOR ERROR CONTROL
C  BASED ON EITHER THE LOCAL VALUE OR THE GLOBAL MAXIMUM OF EACH
C  COMPONENT.
C
C  THE OPTION FOR ERROR CONTROL ON AN ERROR PER UNIT-TIME-STEP OR
C  ERROR PER TIME-STEP BASIS IS ALSO PROVIDED.
C
C  INPUT
C
C    X       - X=X(T), THE APPROXIMATE SOLUTION FOR WHICH AN ERROR
C              CRITERION IS DESIRED.
C    NX      - THE LENGTH OF THE SOLUTION VECTOR X.
C    T       - CURRENT VALUE OF THE TIME VARIABLE.
C    DT      - CURRENT TIME-STEP.
C    ERRPAR  - TWO PARAMETERS FOR USE IN DETERMINING THE DESIRED ERROR.
C    ERPUTS  - IF ERPUTS=.TRUE., THEN THE ERROR IS TO BE
C              PROPORTIONAL TO DABS(DT). OTHERWISE IT WILL NOT.
C    E       - X(I) IS ACCURATE TO A REAL ABSOLUTE ERROR OF E(I),
C              I=1,...,NX, FOR THE SINGLE CURRENT TIME-STEP.
C
C  COMMON INPUT
C
C    IGMAX   - THE POINTER TO THE REAL VECTOR OF CURRENT MAXIMUM ABSOLUTE
C              VALUES ATTAINED BY EACH COMPONENT OF THE SOLUTION.
C              IGMAX=0 MEANS THIS VECTOR IS NOT USED AND HAS NOT BEEN
C              ALLOCATED.
C
C  OUTPUT
C
C    E       - THE REAL ERROR VECTOR. E(I) IS THE ABSOLUTE ERROR
C              TOLERABLE IN X(I), FOR I=1,...,NX.
C
C              LET V(I) = ABS(X(I)) IF IGMAX=0
C                           OTHERWISE
C                       = MAXIMUM(ABS(X(I)(T))) OVER ALL PREVIOUS TIME.
C                         THIS VALUE IS STORED IN THE REAL STACK
C                         POSITION RS(I+IGMAX-1).
C
C              AND EPS = 1 IF ERPUTS=.FALSE.
C                          OTHERWISE
C                      = DABS(DT),
C
C              THEN
C
C                       E(I) = EPS * (ERRPAR(1)*V(I) + ERRPAR(2)),
C
C              FOR I=1,...,NX.
C
C  FUNCTION VALUE
C
C    DODESE  - .TRUE. IF EACH X(I) IS ACCURATE TO WITHIN AN
```

```
C              ABSOLUTE ERROR OF E(I), I=1,...,NX, OTHERWISE .FALSE. .
C
C  SCRATCH SPACE ALLOCATED - NONE.
C
C  ERROR STATES - NONE.
C
      COMMON /DODESM/IGMAX,IGMAXO
C
      DOUBLE PRECISION X(NX),T,DT
      REAL ERRPAR(2),E(NX)
      LOGICAL ERPUTS
C
      COMMON /CSTAK/S
      DOUBLE PRECISION S(500)
      REAL RS(1000),DTPOW,TEMP
      EQUIVALENCE (S(1),RS(1))
C
      DTPOW=1.0E0
      IF (ERPUTS) DTPOW=DABS(DT)
C
      DODESE=.TRUE.
      J=IGMAX
C
      DO 10 I=1,NX
C
         IF (IGMAX.GT.0) TEMP=RS(J)
         IF (IGMAX.EQ.0) TEMP=ABS(SNGL(X(I)))
         TEMP=DTPOW*(ERRPAR(1)*TEMP+ERRPAR(2))
C
         IF (E(I).GT.TEMP) DODESE=.FALSE.
C
         E(I)=TEMP
C
 10      J=J+1
C
      RETURN
C
      END
```

A very simple-minded default HANDLE subroutine DODESH is provided with DODES. It simply returns at the end of each time-step. It is listed below as an example of what a user supplied HANDLE subroutine might look like.

```
      SUBROUTINE DODESH(T0,X0,T1,X1,NX,DT,TSTOP,E)
C
C  THE DEFAULT OUTPUT ROUTINE FOR USE WITH DODES.
C  IT SIMPLY RETURNS.
C
C  SCRATCH SPACE ALLOCATED - NONE.
C
C  ERROR STATES - NONE.
C
      DOUBLE PRECISION T0,X0(NX),T1,X1(NX),DT,TSTOP
```

```
      REAL E(NX)
C
      RETURN
C
      END
```

Figure 1 gives a picture of the basic flow of control through DODES and the user supplied subprograms. DODES calls DODES1 with ERROR=DODESE, GLBMAX=.FALSE. and ERPUTS=.FALSE.. DODES1 calls DODES2 with KMAX=10 and MMAX=16.

4. Examples

This section provides several examples of the use of the subprograms in DODES. The system studied is

$$\begin{aligned} x'_1 &= -x_1 \\ x'_2 &= x_3 \\ x'_3 &= -x_2 \end{aligned} \tag{4.1}$$

on [0,10], subject to

$$\begin{aligned} x_1(0) &= 1 \\ x_2(0) &= 0 \\ x_3(0) &= 1. \end{aligned} \tag{4.2}$$

The solution of (4.1) subject to (4.2) is,

$$\begin{aligned} x_1(t) &= e^{-t} \\ x_2(t) &= \sin(t) \\ x_3(t) &= \cos(t). \end{aligned} \tag{4.3}$$

Thus, we know the exact solution of the problem and can compare the results DODES gives with it.

The simplest way to solve (4.1) subject to (4.2) would be to code the main program

```
      DOUBLE PRECISION X(3),DT
      REAL ERRPAR(2)
      EXTERNAL F123,DODESH
C
C COMPUTE THE SOLUTION TO AN ABSOLUTE ERROR OF 10**(-6).
C
      ERRPAR(1)=0.0D0
      ERRPAR(2)=1.0D-6
C
C DT MUST BE A VARIABLE SINCE DODES WILL ALTER IT.
C
      DT=1.0D-2
C
C SET UP THE INITIAL CONDITIONS
C
      X(1)=1.0D0
      X(2)=0.0D0
      X(3)=1.0D0
C
```

```
      CALL DODES(F123,X,3,0.0D0,10.0D0,DT,ERRPAR,DODESH)
C
      WRITE(6,9000) (X(I),I=1,3)
 9000 FORMAT(9H X(10) = ,1P3D20.8)
      STOP
      END
```

and the subroutine

```
      SUBROUTINE F123(T,X,NX,FTX)
C
      DOUBLE PRECISION T,X(3),FTX(3)
C
      FTX(1)=-X(1)
      FTX(2)=X(3)
      FTX(3)=-X(2)
C
      RETURN
      END
```

The output of this program unit is

```
 X(10) =        4.53999295D-05      -5.44021120D-01      -8.39071520D-01
```

A skeptical user might next decide to check that the solution is in fact accurate to about 10^{-6}, count how many times the subroutine F123 is called, as a means of measuring the "cost" of using DODES, and also check the claim that the cost of using DODES is substantially independent of the initial value of DT chosen. In that case, the user might write a main program like

```
C
C NFCALL WILL COUNT THE NUMBER OF FUNCTION CALLS.
C
      COMMON /FCOUNT/NFCALL
      DOUBLE PRECISION X(3),DT
      REAL ERRPAR(2)
      EXTERNAL F123,CHKOUT
C
      ERRPAR(1)=0.0D0
      ERRPAR(2)=1.0D-6
C
      DT=10.0D0
C
      DO 10 I=1,2
C
C WHEN I=1, DT=10 AND WHEN I=2, DT=10**(-12).
C
      IF (I.EQ.2) DT=1.0D-12
      WRITE(6,9000) DT
 9000 FORMAT(10H FOR DT = ,1P1D20.8///)
```

```
C
      X(1)=1.0D0
      X(2)=0.0D0
      X(3)=1.0D0
      NFCALL=0
C
 10   CALL DODES(F123,X,3,0.0D0,10.0D0,DT,ERRPAR,CHKOUT)
C
      STOP
      END
```

The subprogram F123 would then be altered to increase NFCALL by one every time it is called. Finally, the output CHKOUT subroutine might look like

```
      SUBROUTINE CHKOUT(T0,X0,T1,X1,NX,DT,TSTOP,E)
C
      COMMON /FCOUNT/NFCALL
      DOUBLE PRECISION T0,X0(3),T1,X1(3),DT,TSTOP
      REAL E(NX)
C
      IF (T0.EQ.T1) RETURN
C
C COMPUTE AND PRINT OUT THE ERROR IN THE SOLUTION X.
C
      ERROR1=X1(1)-DEXP(-T1)
      ERROR2=X1(2)-DSIN(T1)
      ERROR3=X1(3)-DCOS(T1)
C
      WRITE(6,9000) T1,ERROR1,ERROR2,ERROR3
 9000 FORMAT (9H ERRORS( ,1P1D12.3,5H ) = ,1P3D12.3)
C
      IF (T1.EQ.TSTOP) WRITE(6,9001) NFCALL
 9001 FORMAT(1H ///34H THE NUMBER OF FUNCTION CALLS WAS ,I6///)
C
      RETURN
      END
```

The output from this program unit is

```
 FOR DT =        1.00000000D 01


 ERRORS(     1.927D-02 ) =     7.313D-08    7.454D-08   -1.437D-09
 ERRORS(     6.277D-02 ) =     7.007D-08    7.436D-08   -4.675D-09
 ERRORS(     3.596D-01 ) =     5.165D-08    6.946D-08   -2.624D-08
 ERRORS(     1.215D 00 ) =     2.188D-08    2.631D-08   -6.941D-08
 ERRORS(     2.896D 00 ) =     3.838D-09   -7.156D-08   -1.874D-08
 ERRORS(     5.537D 00 ) =     2.342D-10    5.429D-08    5.060D-08
 ERRORS(     9.049D 00 ) =     6.739D-12   -6.883D-08   -2.749D-08
 ERRORS(     1.000D 01 ) =     2.625D-12   -6.263D-08    4.320D-08
```

```
 THE NUMBER OF FUNCTION CALLS WAS     341

 FOR DT =        1.00000000D-12

 ERRORS(     1.000D-12 ) =     7.077D-17    1.972D-30    0.000D-39
 ERRORS(     1.043D-09 ) =     7.061D-17    1.616D-27   -2.168D-19
 ERRORS(     1.086D-06 ) =     7.046D-17    1.332D-20   -4.337D-19
 ERRORS(     1.132D-03 ) =     1.504D-11    1.506D-11   -1.704D-14
 ERRORS(     4.463D-02 ) =     6.771D-11    3.169D-11    4.351D-13
 ERRORS(     3.404D-01 ) =    -3.664D-10   -3.078D-10   -9.089D-12
 ERRORS(     1.193D 00 ) =    -2.414D-10    2.968D-10    4.407D-10
 ERRORS(     2.880D 00 ) =    -2.948D-10    2.873D-10   -2.983D-09
 ERRORS(     5.492D 00 ) =    -5.616D-11   -1.355D-09    2.161D-09
 ERRORS(     8.973D 00 ) =    -1.951D-12    6.315D-10   -2.498D-09
 ERRORS(     1.000D 01 ) =    -5.107D-13   -2.050D-09   -6.479D-08

 THE NUMBER OF FUNCTION CALLS WAS     355
```

The results of this simple test bear out the claim that DODES produces results as accurate as the user has requested and that the cost of obtaining these results is indeed quite insensitive to the initial choice of DT.

The perspicacious reader notes that $x_1(10) = e^{-10}$ which is roughly 10^{-4}. Thus, $x_1(10)$ may not have any good digits when (4.1) is solved to an absolute error of say 10^{-2}. He can easily remedy this problem by coding his own error subprogram to provide for a relative error test on x_1 and an absolute error test for x_2 and x_3.

```
      LOGICAL FUNCTION RAEROR(X,NX,T,DT,ERRPAR,ERPUTS,E)
C
      DOUBLE PRECISION X(3),T,DT
      REAL ERRPAR(2),E(3)
      LOGICAL ERPUTS
C
C RELATIVE ERROR FOR X1, ABSOLUTE ERROR FOR X2 AND X3.
C
      RAEROR=.TRUE.
      TEMP=ERRPAR(1)*DABS(X(1))
      IF (TEMP.LT.E(1)) RAEROR=.FALSE.
      E(1)=TEMP
      DO 10 I=2,3
      IF (E(I).GT.ERRPAR(2)) RAEROR=.FALSE.
 10   E(I)=ERRPAR(2)
C
      RETURN
      END
```

The main program would be altered to read

```
      COMMON /FCOUNT/NFCALL
      DOUBLE PRECISION X(3),DT
      REAL ERRPAR(2)
      EXTERNAL F123,CHKOUT,RAEROR
C
      ERRPAR(1)=1.0D-6
      ERRPAR(2)=1.0D-6
C
      DO 10 I=1,2
C
      IF (I.EQ.1) DT=10.0D0
      IF (I.EQ.2) DT=1.0D-12
C
      WRITE(6,9000) DT
 9000 FORMAT(10H FOR DT = ,1P1D20.8///)
C
      X(1)=1.0D0
      X(2)=0.0D0
      X(3)=1.0D0
      NFCALL=0
C
 10   CALL DODES1(F123,X,3,0.0D0,10.0D0,DT,RAEROR,ERRPAR,CHKOUT,
     1            .FALSE.,.FALSE.)
C
      STOP
      END
```

and the output subroutine CHKOUT would be altered to print out the relative error in x_1 rather than the absolute error. The output from this program unit is then

```
 FOR DT =        1.00000000D 01


 ERRORS(     1.927D-02 ) =     7.455D-08    7.454D-08   -1.437D-09
 ERRORS(     6.277D-02 ) =     7.461D-08    7.436D-08   -4.675D-09
 ERRORS(     3.559D-01 ) =     7.406D-08    6.958D-08   -2.598D-08
 ERRORS(     1.141D 00 ) =     7.385D-08    3.110D-08   -6.745D-08
 ERRORS(     2.690D 00 ) =     7.249D-08   -7.115D-08   -3.308D-08
 ERRORS(     4.445D 00 ) =     6.465D-08   -2.112D-08    7.711D-08
 ERRORS(     7.000D 00 ) =     5.883D-08    6.004D-08   -5.279D-08
 ERRORS(     1.000D 01 ) =     5.873D-08   -6.690D-08    4.379D-08


 THE NUMBER OF FUNCTION CALLS WAS     357


 FOR DT =        1.00000000D-12


 ERRORS(     1.000D-12 ) =     7.080D-17    1.972D-30    0.000D-39
 ERRORS(     1.043D-09 ) =     7.069D-17    1.616D-27   -2.168D-19
 ERRORS(     1.086D-06 ) =     7.047D-17    1.332D-20   -4.337D-19
```

```
ERRORS(     1.132D-03 ) =     1.506D-11    1.506D-11   -1.704D-14
ERRORS(     4.463D-02 ) =     7.080D-11   -3.169D-11    4.351D-13
ERRORS(     3.377D-01 ) =     4.756D-10   -2.909D-10   -7.893D-12
ERRORS(     1.122D 00 ) =     6.890D-10    9.083D-12    2.946D-10
ERRORS(     2.675D 00 ) =     2.093D-09    8.637D-10   -3.840D-10
ERRORS(     4.422D 00 ) =     9.452D-09   -1.643D-09    9.480D-10
ERRORS(     6.985D 00 ) =     1.555D-08    2.275D-09   -1.401D-10
ERRORS(     1.000D 01 ) =     1.566D-08   -2.281D-09   -1.230D-10

THE NUMBER OF FUNCTION CALLS WAS     371
```

These examples illustrate the ways in which the user may control both the accuracy of the computed solution and the way in which the computed solution is used, saved, printed, etc.

The subroutine DODES2 provides control over the order of the integration method used. If the user wants to solve a large, say NX=1000, system of ordinary differential equations and he uses DODES to solve it in DOUBLE PRECISION, then roughly 18,000 DOUBLE PRECISION words of memory will be allocated in the stack, see (3.1). This is because DODES indirectly calls DODES2 with KMAX=10 and MMAX=16. If, however, the user only wants the solution accurate to 1%, then it is pretty clear that KMAX=10 (up to a 20^{th} order method) is not needed. In fact, KMAX=3 (a 6^{th} order method) is probably quite sufficient. If he uses DODES2 with KMAX=3 and MMAX=9, then only about 7,500 DOUBLE PRECISION words will be used in the stack, see (3.6). Section 5 shows how the user can easily find out what maximal order is being used by DODES during the integration process. Thus, after one run, the user can set KMAX and MMAX appropriately.

5. Implementation Details

This section discusses various details of the implementation of DODES by which the user with sophisticated needs may access the integration process.

A rare, but important, problem arises when the function $f(t,x)$ cannot be evaluated for some values of t and x. For example, the equation $x'=x^{1/2}$ runs into real trouble if x ever goes negative. This can happen because the solution is typically not computed very accurately by Gragg's modified mid-point rule, the algorithm relying on extrapolation to provide accurate answers. Thus, some mechanism must be provided for the user to say that he cannot calculate $f(t,x)$. This is accomplished as follows: A labelled common region

COMMON /DODESF/OKAY

contains the LOGICAL variable OKAY which indicates whether or not the call to F(T,X,NX,FTX) was successful. A subroutine D0DESG implements Gragg's modified mid-point rule and before each call to F, that routine sets OKAY=.TRUE. . (The subroutine's name D0DESG follows the PORT library convention of using names, for seldom used global variables, which have a digit as their second character, to avoid conflicts with user variables.) Thus, the user need only set OKAY (=.FALSE.) when he cannot evaluate $f(t,x)$. When D0DESG detects OKAY=.FALSE. upon return from a call to F, it returns to its caller, the step-size and order monitor DSXTRP [14]. The default response of DSXTRP is to lower DT by a factor of 10^3 and do a few other reasonable things. In HANDLE, the user may elect to override that default action by simply lowering DT himself. However, this leads to an 8^{th} error state in DODES - which an alert reader may have noticed was omitted between error states 7

and 9 in section 3 - The subroutine HANDLE cannot raise DT when OKAY=.FALSE. .

The following subroutine illustrates the use of OKAY when trying to solve $x'=x^{1/2}$.

```
      SUBROUTINE F(T,X,NX,FTX)
C
      COMMON /DODESF/OKAY
      DOUBLE PRECISION T,X(NX),FTX(NX)
      LOGICAL OKAY
C
      IF (X(1).GE.0.0D0) FTX(1)=DSQRT(X(1))
      IF (X(1).LT.0.0D0) OKAY=.FALSE.
C
      RETURN
      END
```

If GLBMAX=.TRUE., then the maximum absolute value of each component of the solution is to be recorded by DODES. This information is stored in the labelled common region

COMMON /DODESM/IGMAX,IGMAXO

in the form of pointers IGMAX and IGMAXO into the stack [9]. IGMAX is the pointer to the REAL vector, of length NX, of current maximum absolute values attained by each component of the solution since the start of the integration procedure at time TSTART. That is, the maximum absolute value of $x_I(t)$ achieved so far during the run is stored in the REAL stack at location RS(IGMAX+I-1). IGMAXO is the pointer to the REAL vector, of length NX, of maximum absolute values attained by each component of the solution, as of the previous time-step. IGMAX=0 means that GLBMAX=.FALSE. and these vectors are not to be used and have not been allocated in the stack.

The monitor DSXTRP [14] provides a labelled COMMON region

COMMON /D9XTRP/ICOM(9)

which contains information about the current status of the integration process and also provides a means for altering the way DSXTRP does its job. The truly ambitious reader may consult [14] for the details of what this common region contains and what can be done through it. The first two elements of this region are useful in following what order process is being used by the step-size and order monitor DSXTRP. ICOM(1) is the current level of extrapolation, that is, the number of entries already computed in the first column of the extrapolation lozenge. ICOM(2) is the size (number of columns) of the optimal lozenge as predicted for the next time-step. ICOM(1) is defined for use by any of the user subprograms F, ERROR and HANDLE. However, ICOM(2) is only defined for use by HANDLE.

When using a "small" value of KMAX, it would be wise to monitor the maximum value of ICOM(2) (the number of columns in the "optimal" size lozenge) used during the integration process. If that value is equal to KMAX, then KMAX should be increased for subsequent runs with similar input data. This should lower the cost of these later runs.

Bibliography

[1] R. Bulirsch and J. Stoer, "Fehlerabschatzungen und Extrapolation mit rationalen Funktionen bei Verfahren vom Richardson-Typus", Num. Math. 6, 413-427 (1964).

[2] R. Bulirsch and J. Stoer, "Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods", Num. Math. 8, 1-13 (1966).

[3] P. A. Fox, "DESUB : Integration of a First Order System of Ordinary Differential Equations", **Mathematical Software**, (Ed. J.R. Rice), Academic Press, pp477-507,1971.

[4] P.A. Fox, A.D. Hall and N.L. Schryer, "Machine Constants for the PORT Library", to appear.

[5] P. A. Fox, "A Comparative Study of Computer Programs for Integrating Differential Equations", Comm. ACM 15, 941-948 (1972).

[6] P. A. Fox, The PORT Library Project, to appear.

[7] C. W. Gear, "The Automatic Integration of Ordinary Differential Equations", Comm. ACM 14, 176-179 (1971).

[8] W. B. Gragg, "Repeated Extrapolation to the Limit in the Numerical Solution of Ordinary Differential Equations", Thesis, UCLA (1963).

[9] A.D. Hall and N.L. Schryer, "A Portable Dynamic Storage Allocator for FORTRAN Programs", to appear.

[10] A.D. Hall and N.L. Schryer, "A Centralized Error Handling Facility for Portable FORTRAN Libraries", to appear.

[11] T.E. Hull, W.H. Enright, B.M. Fellen and A.E. Sedgwick, "Comparing Numerical Methods for Ordinary Differential Equations", SIAM J. Numer. Anal. 9, 603-637(1972).

[12] F.T. Krogh, "On Testing a Subroutine for the Numerical Integration of Ordinary Differential Equations", J. of the ACM 20, 545-562(1973).

[13] B.G. Ryder, "The PFORT Verifier: User's Guide", Computing Science Technical Report #12, Bell Laboratories, Murray Hill, N.J., 1973; rev. 1975.

[14] N.L. Schryer, "An Extrapolation Step-Size and Order Monitor for Use in Solving Ordinary Differential Equations", Proceedings ACM National Meeting, November, 1974.

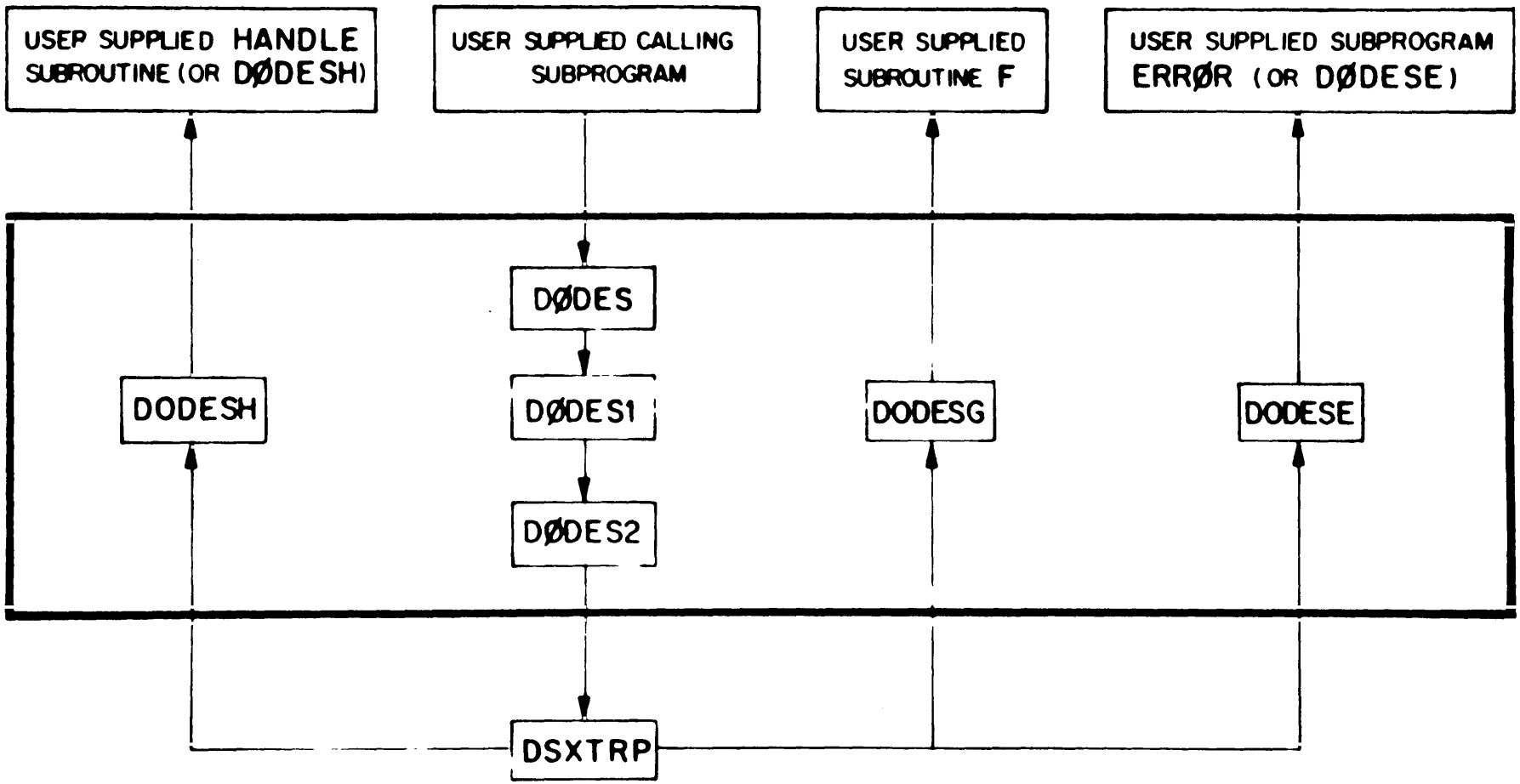
[15] N.L. Schryer, "An Extrapolation Step-Size and Order Monitor for Use in Solving Ordinary Differential Equations", to appear.

[16] H.J. Stetter, "Asymptotic Expansions for the Error of Discretization Algorithms for Non-Linear Functional Equations", Num. Math. 7, 18-31 (1965).

[17] J. Stoer, "Extrapolation Methods for the Solution of Initial Value Problems and their Practical Realization", Conference on the Numerical Solution of Ordinary Differential Equations, University of Texas at Austin, 1972.

[18] D.D. Warner, A Stiff Differential Equation Solver, in preparation.

[19] T. Yu, "Comparision of Numerical Methods for Ordinary Differential Equations", Technical Report CNA-73, 1973, University of Texas at Austin.



ALL SUBPROGRAMS IN THE HEAVY BOX ARE IN THE DØDES PACKAGE

FIGURE 1