Warning: Subroutine `DA01` has been superseded by subroutine `DC03` which uses improved algorithms; the use of the latter routine is recommended. The superseded routine may be removed from later releases of the library.

1 SUMMARY

To integrate a set of **first order ordinary differential equations**

$$y_i' = f_i(y_1, y_2, .., y_n, x) \quad i=1,2,...,n$$

given initial conditions $y_i(x_0) = y_i^0$ *at* x_0 .

Each call to the subroutine advances the integration one step from the current point x to $x+\delta x$ where δx the steplength must be specified by the user. The 4 th order Runge-Kutta method due to Merson is used which attempts to estimate the truncation errors for the step. To integrate over a range $x_0 \le x \le x_e$ the user must write a driver program which repeatedly calls the subroutine until the range is covered. If the solution is required within a certain accuracy it is the responsibility of the user to control the steplength so that both accuracy and stability are achieved. The Merson error estimates can be used for this.

The user must provide a subroutine to compute values of the derivative functions $f_i(y_1, y_2, .., y_n, x) \quad i=1,2,...,n$.

ATTRIBUTES — **Version:** 1.0.0. **Remark:** See also `DA02A/AD` which provides automatic steplength control for `DA01A/AD` and also `DC03AD` and `DC04AD` which have been developed specially to integrate so-called ‘stiff’ systems of equations. Recent comparisons suggest that `DC03AD` may be much more efficient on many types of problem, specially if derivative calculations are expensive. **Types:** `DA01A; DA01AD.` **Calls:** `DYBDX` (user subroutine). **Original date:** March 1963. **Origin:** D.McVicar, Harwell.

2 HOW TO USE THE PACKAGE

2.1 The argument list and calling sequence

The single precision version:

```
CALL DA01A(Y,E,W1,W2,W3,W4,N,X,DX)
```

The double precision version:

```
CALL DA01AD(Y,E,W1,W2,W3,W4,N,X,DX)
```

`Y` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length at least n, which must be set by the user to the current values of y_i $i=1,2,...,n$, i.e. the y values corresponding to the point x, and on return will have been set by the subroutine to the updated values, i.e. those for the point $x+\delta x$.

Initially the user must set the array to the initial values y_i^0 $i=1,2,...,n$. If re-starts with a smaller steplength are likely, arrangements must be made to preserve the current values between calls, see §2.3 on controlling the steplength.

`E` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length at least n, which will be set by the subroutine to the truncation error estimates e_i $i=1,2,...,n$ for the step just taken. These may be used to adjust the steplength, see §4.

`W1, W2, W3, W4` are four `REAL` (`DOUBLE PRECISION` in the D version) arrays each of length at least n words which will be used by the subroutine as workspace.

`N` is an `INTEGER` variable which must be set by the user to n the number of equations, (If n=1 the arguments `Y, E, W1, W2, W3` and `W4` need not be arrays).

`X` is a `REAL` (`DOUBLE PRECISION` in the D version) variable and contains the current value of the point reached in the integration. On entry it must contain the value x and on return it will have been set to $x+\delta x$. Initially the user must set `X` to the initial point x_0.

`DX` is a `REAL` (`DOUBLE PRECISION` in the D version) variable which must be set by the user to the steplength δx . This argument is not altered by the subroutine. A suggestion for an automatic steplength control scheme is outlined in §2.3.

2.2 Computing values of the derivative functions.

The user must provide a subroutine called `DYBDX` to compute values of the derivative functions $f_i(y_1,y_2,..,y_n,x)$ i=1,2,...,n.

The subroutine definition for both single and double precision versions should be

```
SUBROUTINE DYBDX(Y,F,N,X)
DIMENSION Y(N),F(N)
   --
   --
RETURN
END
```

`Y` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length at least n containing the values of y_i i=1,2,...,n.

`F` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length at least n which the user must use to return the values of the functions $f_i(y_1,y_2,..,y_n,x)$ i=1,2,...,n .

`N` is an `INTEGER` variable and will give n the number of functions.

`X` is a `REAL` (`DOUBLE PRECISION` in the D version) variable and will give the value of x .

The user must not alter the contents of `Y,N` or `X.` At each step in the integration the subroutine `DYBDX` is called 5 times by `DA01A/AD` (note that the name `DYBDX` is used by both the single and double precision versions of `DA01` and the word lengths of its arguments must match those of the version being used).

Any additional information that `DYBDX` may require can be passed to it using Common or by giving it a second entry point.

2.3 Controlling the steplength.

If all the error estimates $|e_i|$ i=1,2,...,n , (e_i is returned in `E(i), I=1,N`) satisfy the user's criteria of accuracy the subroutine may be re-entered without need to re-set any of the arguments. If there is any risk that the steplength δx is too large for the required accuracy, arrangements must be made to preserve at each integration step the current values of x and y_i i=1,2,...,n before calling the subroutine so that they may be restored together with a smaller value of δx for a retry.

If at least one $|e_i|$ needs to be reduced by a factor α, the new δx should be approximately $\alpha^{0.2}$ times the old one. If on the other hand all $|e_i|$ could be increased by a factor α and still be acceptable, δx could also be increased to a value $\alpha^{0.2}$ times its old value. This is illustrated in the examples of section 7, there the steplength is adjusted by the formula

$$\delta x_{new} = \delta x_{old}(0.9\alpha^{-0.2})$$

where an extra factor 0.9 is applied. Also in practice it is a good thing to impose limits on the extent to which the steplength may change at any one time.

3 GENERAL INFORMATION

Use of Common: none.

Workspace: all supplied by the user in the arrays `W1, W2, W3` and `W4,` each of length n .

Other subroutines: the user must supply a subroutine called `DYBDX` to compute values of the derivative functions.

Input/Output: none.

4 METHOD

The method used is the 4 th order Runge-Kutta method due to R.H. Merson, see (1947) Proc. of Symp. on Data Processing, W.R.E., South Australia. In this method 5 th order terms are used to estimate truncation errors.

5 EXAMPLE OF USE

Suppose we are to solve the second order equation

$$y'' = e^{-x} - \theta(y' + y^3)$$

for certain values of the parameter θ with initial conditions y=1 , y’ = –1 at x=0 . To use a first order method the equation is transformed into a pair of first order equations; this is done by defining $y_1 = y' \, and \, y_2 = y$ and thus obtaining

$y_1' = e^{-x} - \theta(y_1 + y_2^3)$

$y_2' = y_1$

with initial conditions $y_1^0 = -1 \, and \, y_2^0 = 1 \, at \, x_0 = 0$.

We propose to integrate out to *x*=2 and solve for various values of θ and for various accuracies. These we shall read from data cards. The steplength control outlined in §2.3 will be used and we shall start with an initial steplength equal to a tenth of the range.

In addition to the driver program we need a subroutine to compute values of the derivative functions. This will require the value of θ and we pass this through the Common area called `PAR.`

N.B. In practice such a program as outlined above is likely to contain more sophistications than the one in this example. An example like this attempts only to illustrate, in a simple way, ways of solving some of the programming problems which arise when using integrators such as `DA01A/AD` and to reiterate points made elsewhere in the write up.

The example code follows.

```
C     y  values, save  y values, error est.
      REAL Y(2),YR(2),E(2)
C     work arrays for DA01A
      REAL W1(2),W2(2),W3(2),W4(2)
C     to pass  theta  to DYBDX
      COMMON/PAR/THETA
C     case no., theta , accuracy requirement
    1 READ(5,2,END=10) ICASE,THETA,EPS
    2 FORMAT(I5,F10.0,E10.2)
      X=0.
C     set initial values
      Y(1)=-1.
      Y(2)=1.
C     end of range of x
      XEND=2.
C     initial steplength
```

```
      DX=(XEND-X)*.1
      GO TO 5
C     prevent delta increasing too much
    3 ALPHA=AMAX1(ALPHA,1E-5)
C     compute new  delta x
      DX=DX*.9*ALPHA**(-.2)
C     make sure we don't go beyond the end of the range
    4 IF(X+DX.LE.XEND) GO TO 5
      DX=XEND-X
C     test for integration complete
      IF(DX.LE.0.) GO TO 8
    5 YR(1)=Y(1)
C     save current values in case a retry is necessary
      YR(2)=Y(2)
      XR=X
C     integrate forward one step
    6 CALL DA01A(Y,E,W1,W2,W3,W4,2,X,DX)
C     test for acceptance of the errors
      ALPHA=AMAX1(ABS(E(1)),ABS(E(2)))/EPS
      IF(ALPHA.LE.1.) GO TO 3
      Y(1)=YR(1)
C     errors not accepted restore old values ready to try again with
      smaller  delta x
      Y(2)=YR(2)
      X=XR
C     limit reduction factor and reduce steplength
      ALPHA=AMIN1(ALPHA,1E8)
      DX=DX*.9*ALPHA**(-.2)
C     is delta x still significant?
      IF(X+DX.NE.X) GO TO 6
C     no: print error diagnostic
      WRITE(6,7) ICASE
    7 FORMAT('ERROR: CASE NO. ',I2,'TOO HIGH AN ACCURACY REQUESTED')
      GO TO 1
C     print final values
    8 WRITE(6,9) ICASE,Y(2),THETA
    9 FORMAT('CASE NO. ',I2,' SOLUTION = ',E13.6,' THETA = ',f10.6)
      GO TO 1
   10 STOP"
      END
      SUBROUTINE DYBDX(Y,F,N,X)
C     note: dimensions are dummy
      REAL Y(*),F(*)
C     value of  theta
      COMMON/PAR/THETA
C     value of  y1'
      F(1)=EXP(-X)-THETA*(Y(1)+Y(2)**3)
C     value of  y2'
      F(2)=Y(1)
      RETURN
      END
```