NAG Library Routine Document

D02LAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D02LAF is a routine for integrating a non-stiff system of second-order ordinary differential equations using Runge–Kutta–Nystrom techniques.

2 Specification

```
SUBROUTINE D02LAF (FCN, NEQ, T, TEND, Y, YP, YDP, RWORK, LRWORK, IFAIL)
INTEGER             NEQ, LRWORK, IFAIL
REAL (KIND=nag_wp) T, TEND, Y(NEQ), YP(NEQ), YDP(NEQ), RWORK(LRWORK)
EXTERNAL            FCN
```

3 Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathrm{NEQ}}, y'_1, y'_2, \ldots, y'_{\mathrm{NEQ}}$ D02LAF integrates a non-stiff system of second-order differential equations of the type

$$y''_i = f_i\big(x, y_1, y_2, \ldots, y_{\mathrm{NEQ}}\big), \quad i = 1, 2, \ldots, \mathrm{NEQ},$$

from $x =$ T to $x =$ TEND using a Runge–Kutta–Nystrom formula pair. The system is defined by FCN, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_{\mathrm{NEQ}}$, where $y_1, y_2, \ldots, y_{\mathrm{NEQ}}$ are supplied at x.

There are two Runge–Kutta–Nystrom formula pairs implemented in this routine. The lower order method is intended if you have moderate accuracy requirements and may be used in conjunction with the interpolation routine D02LZF to produce solutions and derivatives at user-specified points. The higher order method is intended if you have high accuracy requirements.

In one-step mode the routine returns approximations to the solution, derivative and f_i at each integration point. In interval mode these values are returned at the end of the integration range. You select the order of the method, the mode of operation, the error control and various optional inputs by a prior call to D02LXF.

For a description of the Runge–Kutta–Nystrom formula pairs see Dormand *et al.* (1986a) and Dormand *et al.* (1986b) and for a description of their practical implementation see Brankin *et al.* (1989).

4 References

Brankin R W, Dormand J R, Gladwell I, Prince P J and Seward W L (1989) Algorithm 670: A Runge–Kutta–Nystrom Code *ACM Trans. Math. Software* **15** 31–40

Dormand J R, El–Mikkawy M E A and Prince P J (1986a) Families of Runge–Kutta–Nystrom formulae *Mathematical Report TPMR 86-1* Teesside Polytechnic

Dormand J R, El–Mikkawy M E A and Prince P J (1986b) High order embedded Runge–Kutta–Nystrom formulae *Mathematical Report TPMR 86-2* Teesside Polytechnic

5 Arguments

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (that is the second derivatives y''_i) for given values of its arguments x, $y_1, y_2, \ldots, y_{\mathrm{NEQ}}$.

The specification of FCN is:

```
SUBROUTINE FCN (NEQ, T, Y, F)
INTEGER             NEQ
REAL (KIND=nag_wp) T, Y(NEQ), F(NEQ)
```

1: NEQ – INTEGER *Input*

On entry: the number of differential equations.

2: T – REAL (KIND=nag_wp) *Input*

On entry: x, the value of the argument.

3: Y(NEQ) – REAL (KIND=nag_wp) array *Input*

On entry: y_i, for $i = 1, 2, \ldots, \text{NEQ}$, the value of the argument.

4: F(NEQ) – REAL (KIND=nag_wp) array *Output*

On exit: the value of f_i, for $i = 1, 2, \ldots, \text{NEQ}$.

FCN must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D02LAF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

2: NEQ – INTEGER *Input*

On entry: the number of second-order ordinary differential equations to be solved by D02LAF. It must contain the same value as the argument NEQ used in a prior call to D02LXF.

Constraint: NEQ ≥ 1.

3: T – REAL (KIND=nag_wp) *Input/Output*

On entry: the initial value of the independent variable x.

Constraint: T $\neq$ TEND.

On exit: the value of the independent variable, which is usually TEND, unless an error has occurred or the code is operating in one-step mode. If the integration is to be continued, possibly with a new value for TEND, T must not be changed.

4: TEND – REAL (KIND=nag_wp) *Input*

On entry: the end point of the range of integration. If TEND $<$ T on initial entry, integration will proceed in the negative direction. TEND may be reset, in the direction of integration, before any continuation call.

5: Y(NEQ) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the initial values of the solution $y_1, y_2, \ldots, y_{\text{NEQ}}$.

On exit: the computed values of the solution at the exit value of T. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

6: YP(NEQ) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the initial values of the derivatives $y'_1, y'_2, \ldots, y'_{\text{NEQ}}$.

On exit: the computed values of the derivatives at the exit value of T. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

7: YDP(NEQ) – REAL (KIND=nag_wp) array *Input/Output*

On entry: must be unchanged from a previous call to D02LAF.

On exit: the computed values of the second derivative at the exit value of T, unless illegal input is detected, in which case the elements of YDP may not have been initialized. If the integration is to be continued, possibly with a new value for TEND, these values must not be changed.

8: RWORK(LRWORK) – REAL (KIND=nag_wp) array *Communication Array*

This **must** be the same argument RWORK as supplied to D02LXF. It is used to pass information from D02LXF to D02LAF, and from D02LAF to both D02LYF and D02LZF. Therefore the contents of this array **must not** be changed before the call to D02LAF or calling either of the routines D02LYF and D02LZF.

9: LRWORK – INTEGER *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02LAF is called.

This must be the same argument LRWORK as supplied to D02LXF.

10: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, because for this routine the values of the output arguments may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

Illegal input detected, i.e., one of the following conditions:

on any call, T = TEND, or the value of NEQ or LRWORK has been altered;

on a continuation call, the direction of integration has been changed;

D02LXF had not been called previously, or the previous call to D02LXF resulted in an error exit.

This error exit can be caused if elements of RWORK have been overwritten.

IFAIL $= 2$

The maximum number of steps has been attempted. (See argument MAXSTP in D02LXF.) If integration is to be continued then you need only reset IFAIL and call the routine again and a further MAXSTP steps will be attempted.

IFAIL $= 3$

In order to satisfy the error requirements, the step size needed is too small for the ***machine precision*** being used.

IFAIL $= 4$

The code has detected two successive error exits at the current value of x and cannot proceed. Check all input variables.

IFAIL $= 5$

The code has detected inefficient use of the integration method. The step size has been reduced by a significant amount too often in order to hit the output points specified by TEND. (Of the last 100 or more successful steps more than 10% are steps with sizes that have had to be reduced by a factor of greater than a half.)

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

The accuracy of integration is determined by the arguments TOL, THRES and THRESP in a prior call to D02LXF. Note that only the local error at each step is controlled by these arguments. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the system. The code is designed so that a reduction in TOL should lead to an approximately proportional reduction in the error. You are strongly recommended to call D02LAF with more than one value for TOL and to compare the results obtained to estimate their accuracy.

The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. For a description of the error test see the specifications of the arguments TOL, THRES and THRESP in routine document D02LXF.

8 Parallelism and Performance

D02LAF is not thread safe and should not be called from a multithreaded user program. Please see Section 3.12.1 in How to Use the NAG Library and its Documentation for more information on thread safety.

D02LAF is not threaded in any implementation.

9 Further Comments

If D02LAF fails with IFAIL $= 3$ then the value of TOL may be so small that a solution cannot be obtained, in which case the routine should be called again with a larger value for TOL. If the accuracy requested is really needed then you should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. D02LAF could be used in one-step mode to monitor the size of the solution with the aim of

trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) if the solution contains fast oscillatory components, the routine will require a very small step size to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with IFAIL = 3. The Runge–Kutta–Nystrom methods are not efficient in such cases and you should consider reposing your problem as a system of first-order ordinary differential equations and then using a routine from Sub-chapter D02M–N with the Blend formulae (see D02MVF).

D02LAF can be used for producing results at short intervals (for example, for tabulation), in two ways. By far the less efficient is to call D02LAF successively over short intervals, $t + (i-1) \times h$ to $t + i \times h$, although this is the only way if the higher order method has been selected and precisely **not** what it is intended for. A more efficient way, **only** for use when the lower order method has been selected, is to use D02LAF in one-step mode. The output values of arguments Y, YP, YDP, T and RWORK are set correctly for a call to D02LZF to compute the solution and derivative at the required points.

10 Example

This example solves the following system (the two body problem)

$$\begin{aligned} y_1'' &= -y_1/\left(y_1^2+y_2^2\right)^{3/2} \\ y_2'' &= -y_2/\left(y_1^2+y_2^2\right)^{3/2} \end{aligned}$$

over the range $[0, 20]$ with initial conditions $y_1 = 1.0 - \epsilon$, $y_2 = 0.0$, $y_1' = 0.0$ and $y_2' = \sqrt{\left(\frac{1+\epsilon}{1-\epsilon}\right)}$ where ϵ, the eccentricity, is 0.5. The system is solved using the lower order method with relative local error tolerances 1.0E−4 and 1.0E−5 and default threshold tolerances. D02LAF is used in one-step mode (ONESTP = .TRUE.) and D02LZF provides solution values at intervals of 2.0.

10.1 Program Text

```
!   D02LAF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d02lafe_mod

!     D02LAF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                             :: fcn
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Integer, Parameter, Public         :: neq = 2, nin = 5, nout = 6
      Integer, Parameter, Public         :: lrwork = 16 + 20*neq
    Contains
      Subroutine fcn(neq,t,y,f)

!       Derivatives for two body problem in  y'' = f(t,y)  form

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)            :: neq
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: f(neq)
        Real (Kind=nag_wp), Intent (In) :: y(neq)
!       .. Local Scalars ..
        Real (Kind=nag_wp)              :: r
!       .. Intrinsic Procedures ..
        Intrinsic                       :: sqrt
```

```
!       .. Executable Statements ..
        r = sqrt(y(1)**2+y(2)**2)**3
        f(1) = -y(1)/r
        f(2) = -y(2)/r
        Return
      End Subroutine fcn
    End Module d02lafe_mod

    Program d02lafe

!     D02LAF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02laf, d02lxf, d02lyf, d02lzf, nag_wp
      Use d02lafe_mod, Only: fcn, lrwork, neq, nin, nout, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: h, hnext, hstart, hused, t, tend,    &
                                         tinc, tnext, tol, tstart
      Integer                         :: i, ifail, itol, maxstp, natt, nfail, &
                                         nsucc, nwant
      Logical                         :: high, onestp, start
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: rwork(:), thres(:), thresp(:), y(:), &
                                         ydp(:), yinit(:), yp(:), ypinit(:),  &
                                         ypwant(:), ywant(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02LAF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
!     neq: number of second-order ordinary differential equations
      Read (nin,*) nwant
      Allocate (rwork(lrwork),thres(neq),thresp(neq),y(neq),ydp(neq),          &
        yinit(neq),yp(neq),ypinit(neq),ypwant(nwant),ywant(nwant))
      Read (nin,*) high, onestp
      Read (nin,*) tinc

!     Initial conditions
      Read (nin,*) tstart, tend
      Read (nin,*) yinit(1:neq)
      Read (nin,*) ypinit(1:neq)

loop1: Do itol = 4, 5
        tol = 10.0_nag_wp**(-itol)
        Write (nout,*)

!       Call D02LXF with default THRES,THRESP,MAXSTP and H

        thres(1) = zero
        thresp(1) = zero
        h = zero
        maxstp = 0
        start = .True.

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d02lxf(neq,h,tol,thres,thresp,maxstp,start,onestp,high,rwork,     &
          lrwork,ifail)

        Write (nout,99999) 'Calculation with TOL = ', tol
        Write (nout,99995)(i,i=1,neq)

!       Set initial values

        y(1:neq) = yinit(1:neq)
        yp(1:neq) = ypinit(1:neq)
        t = tstart
        tnext = t + tinc
        Write (nout,99998) t, y(1:neq)
```

```
!       Loop point for one-step mode
loop2:  Do

          ifail = -1
          Call d02laf(fcn,neq,t,tend,y,yp,ydp,rwork,lrwork,ifail)

          If (ifail>0) Then
            Write (nout,99997) ifail, t
            Exit loop1
          End If

!         Loop point for interpolation
          Do While (tnext<=t)

            ifail = 0
            Call d02lzf(neq,t,y,yp,neq,tnext,ywant,ypwant,rwork,lrwork,ifail)

            Write (nout,99998) tnext, ywant(1:neq)
            tnext = tnext + tinc
          End Do

          If (t>=tend) Then
            Exit loop2
          End If

        End Do loop2

        ifail = 0
        Call d02lyf(neq,hnext,hused,hstart,nsucc,nfail,natt,thres,thresp,      &
          rwork,lrwork,ifail)

        Write (nout,*)
        Write (nout,99996) ' Number of successful steps = ', nsucc
        Write (nout,99996) ' Number of   failed   steps = ', nfail
      End Do loop1

99999 Format (1X,A,1P,E9.1)
99998 Format (1X,F5.1,2(2X,F9.5))
99997 Format (/,1X,'D02LAF returned with IFAIL = ',I2,'  at T = ',1P,E10.3)
99996 Format (1X,A,I5)
99995 Format (/,'   T ',2('     Y(',I1,')  '))
    End Program d02lafe
```

10.2 Program Data

```
D02LAF Example Program Data
   2                            : nwant
   .FALSE. .TRUE.               : high, onestp
   2.0                          : tinc
   0.0  20.0                    : tstart, tend
   0.5  0.0                     : yinit
   0.0  1.73205080756887729352  : ypinit
```

10.3 Program Results

```
 D02LAF Example Program Results

 Calculation with TOL =   1.0E-04

   T      Y(1)       Y(2)
   0.0    0.50000    0.00000
   2.0   -1.20573    0.61357
   4.0   -1.33476   -0.47685
   6.0    0.35748   -0.44558
   8.0   -1.03762    0.73022
  10.0   -1.42617   -0.32658
  12.0    0.05515   -0.72032
  14.0   -0.82880    0.81788
  16.0   -1.48103   -0.16788
```

```
  18.0   -0.26719   -0.84223
  20.0   -0.57803    0.86339

  Number of successful steps =   108
  Number of   failed   steps =    16

 Calculation with TOL =   1.0E-05

   T       Y(1)       Y(2)
   0.0    0.50000    0.00000
   2.0   -1.20573    0.61357
   4.0   -1.33476   -0.47685
   6.0    0.35748   -0.44558
   8.0   -1.03762    0.73022
  10.0   -1.42617   -0.32658
  12.0    0.05516   -0.72031
  14.0   -0.82880    0.81787
  16.0   -1.48103   -0.16789
  18.0   -0.26718   -0.84223
  20.0   -0.57804    0.86338

  Number of successful steps =   169
  Number of   failed   steps =    15
```