NAG Fortran Library Routine Document

D03PRF

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

D03PRF integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs), and automatic adaptive spatial remeshing. The spatial discretization is performed using the Keller box scheme (see Keller (1970)) and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a Backward Differentiation Formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
 SUBROUTINE D03PRF (NPDE, TS, TOUT, PDEDEF, BNDARY, UVINIT, U, NPTS, X,
1                   NLEFT, NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL,
2                   ITOL, NORM, LAOPT, ALGOPT, REMESH, NXFIX, XFIX,
3                   NRMESH, DXMESH, TRMESH, IPMINF, XRATIO, CON, MONITF,
4                   RSAVE, LRSAVE, ISAVE, LISAVE, ITASK, ITRACE, IND,
5                   IFAIL)

 INTEGER            NPDE, NPTS, NLEFT, NCODE, NXI, NEQN, ITOL, NXFIX,
1                   NRMESH, IPMINF, LRSAVE, ISAVE(LISAVE), LISAVE, ITASK,
2                   ITRACE, IND, IFAIL
 double precision   TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*), ATOL(*),
1                   ALGOPT(30), XFIX(*), DXMESH, TRMESH, XRATIO, CON,
2                   RSAVE(LRSAVE)
 LOGICAL            REMESH
 CHARACTER*1        NORM, LAOPT
 EXTERNAL           PDEDEF, BNDARY, UVINIT, ODEDEF, MONITF
```

3 Description

D03PRF integrates the system of first-order PDEs and coupled ODEs given by the master equations:

$$G_i\left(x,t,U,U_x,U_t,V,\dot{V}\right)=0, \quad i=1,2,\ldots,\mathrm{NPDE}, \quad a\leq x\leq b, t\geq t_0, \tag{1}$$

$$F_i\left(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*\right)=0, \quad i=1,2,\ldots,\mathrm{NCODE}. \tag{2}$$

In the PDE part of the problem given by (1), the functions G_i must have the general form

$$G_i=\sum_{j=1}^{\mathrm{NPDE}}P_{i,j}\frac{\partial U_j}{\partial t}+\sum_{j=1}^{\mathrm{NCODE}}Q_{i,j}\dot{V}_j+R_i=0, \quad i=1,2,\ldots,\mathrm{NPDE}, \tag{3}$$

where $P_{i,j}$, $Q_{i,j}$ and R_i depend on x, t, U, U_x and V.

The vector U is the set of PDE solution values

$$U(x,t)=\left[U_1(x,t),\ldots,U_{\mathrm{NPDE}}(x,t)\right]^{\mathrm{T}},$$

and the vector U_x is the partial derivative with respect to x. The vector V is the set of ODE solution values

$$V(t)=\left[V_1(t),\ldots,V_{\mathrm{NCODE}}(t)\right]^{\mathrm{T}},$$

and $\dot{V}$ denotes its derivative with respect to time.

In the ODE part given by (2), ξ represents a vector of n_ξ spatial coupling points at which the ODEs are coupled to the PDEs. These points may or may not be equal to some of the PDE spatial mesh points. U^*, U_x^* and U_t^* are the functions U, U_x and U_t evaluated at these coupling points. Each F_i may only depend linearly on time derivatives. Hence equation (2) may be written more precisely as

$$F = A - B\dot{V} - CU_t^*, \tag{4}$$

where $F = \left[F_1, \ldots, F_{\mathrm{NCODE}}\right]^{\mathrm{T}}$, A is a vector of length NCODE, B is an NCODE by NCODE matrix, C is an NCODE by $\left(n_\xi \times \mathrm{NPDE}\right)$ matrix and the entries in A, B and C may depend on t, ξ, U^*, U_x^* and V. In practice you only need to supply a vector of information to define the ODEs and not the matrices B and C. (See Section 5 for the specification of the user-supplied (sub)program ODEDEF.)

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$ defined initially by you and (possibly) adapted automatically during the integration according to user-specified criteria.

The PDE system which is defined by the functions G_i must be specified in the user-supplied (sub)program PDEDEF.

The initial $(t = t_0)$ values of the functions $U(x,t)$ and $V(t)$ must be specified in a (sub)program UVINIT supplied by you. Note that UVINIT will be called again following any remeshing, and so $U(x, t_0)$ should be specified for **all** values of x in the interval $a \leq x \leq b$, and not just the initial mesh points.

For a first-order system of PDEs, only one boundary condition is required for each PDE component U_i. The NPDE boundary conditions are separated into n_a at the left-hand boundary $x = a$, and n_b at the right-hand boundary $x = b$, such that $n_a + n_b = \mathrm{NPDE}$. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of U_i at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for U_i should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialization or integration difficulties in the underlying time integration routines.

The boundary conditions have the master equation form:

$$G_i^L\left(x, t, U, U_t, V, \dot{V}\right) = 0 \quad \text{at } x = a, \quad i = 1, 2, \ldots, n_a, \tag{5}$$

at the left-hand boundary, and

$$G_i^R\left(x, t, U, U_t, V, \dot{V}\right) = 0 \quad \text{at } x = b, \quad i = 1, 2, \ldots, n_b, \tag{6}$$

at the right-hand boundary.

Note that the functions G_i^L and G_i^R must not depend on U_x, since spatial derivatives are not determined explicitly in the Keller box scheme routines. If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that G_i^L and G_i^R must be linear with respect to time derivatives, so that the boundary conditions have the general form:

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L = 0, \quad i = 1, 2, \ldots, n_a, \tag{7}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^R \dot{V}_j + S_i^R = 0, \quad i = 1, 2, \ldots, n_b, \tag{8}$$

at the right-hand boundary, where $E_{i,j}^L$, $E_{i,j}^R$, $H_{i,j}^L$, $H_{i,j}^R$, S_i^L and S_i^R depend on x, t, U and V only.

The boundary conditions must be specified in a (sub)program BNDARY provided by you.

The problem is subject to the following restrictions:

(i) $P_{i,j}$, $Q_{i,j}$ and R_i must not depend on any time derivatives;

(ii) $t_0 < t_{\mathrm{out}}$, so that integration is in the forward direction;

(iii) The evaluation of the function G_i is done approximately at the mid-points of the mesh X(i), for $i = 1, 2, \ldots,$ NPTS, by calling the user-supplied (sub)program PDEDEF for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the fixed mesh points specified by XFIX;

(iv) At least one of the functions $P_{i,j}$ must be non-zero so that there is a time derivative present in the PDE problem.

The algebraic-differential equation system which is defined by the functions F_i must be specified in the user-supplied (sub)program ODEDEF. You must also specify the coupling points ξ in the array XI.

The first-order equations are approximated by a system of ODEs in time for the values of U_i at mesh points. In this method of lines approach the Keller box scheme is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are NPDE $\times$ NPTS $+$ NCODE ODEs in time direction. This system is then integrated forwards in time using a Backward Differentiation Formula (BDF) or a Theta method.

The adaptive space remeshing can be used to generate meshes that automatically follow the changing time-dependent nature of the solution, generally resulting in a more efficient and accurate solution using fewer mesh points than may be necessary with a fixed uniform or non-uniform mesh. Problems with travelling wavefronts or variable-width boundary layers for example will benefit from using a moving adaptive mesh. The discrete time-step method used here (developed by Furzeland (1984)) automatically creates a new mesh based on the current solution profile at certain time-steps, and the solution is then interpolated onto the new mesh and the integration continues.

The method requires you to supply a (sub)program MONITF which specifies in an analytic or numeric form the particular aspect of the solution behaviour you wish to track. This so-called monitor function is used to choose a mesh which equally distributes the integral of the monitor function over the domain. A typical choice of monitor function is the second space derivative of the solution value at each point (or some combination of the second space derivatives if more than one solution component), which results in refinement in regions where the solution gradient is changing most rapidly.

You must specify the frequency of mesh updates along with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and you are encouraged to experiment with the different options in order to achieve an efficient solution which adequately tracks the desired features of the solution.

Note that unless the monitor function for the initial solution values is zero at all user-specified initial mesh points, a new initial mesh is calculated and adopted according to the user-specified remeshing criteria. The user-supplied (sub)program UVINIT will then be called again to determine the initial solution values at the new mesh points (there is no interpolation at this stage) and the integration proceeds.

4 References

Berzins M (1990) Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (ed J C Mason and M G Cox) 59–72 Chapman and Hall

Berzins M, Dew P M and Furzeland R M (1989) Developing software for time-dependent problems using the method of lines and differential-algebraic integrators *Appl. Numer. Math.* **5** 375–397

Berzins M and Furzeland R M (1992) An adaptive theta method for the solution of stiff and nonstiff differential equations *Appl. Numer. Math.* **9** 1–19

Furzeland R M (1984) The construction of adaptive space meshes *TNER.85.022* Thornton Research Centre, Chester

Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** 327–350 Academic Press

Pennington S V and Berzins M (1994) New NAG Library software for first-order partial differential equations *ACM Trans. Math. Softw.* **20** 63–99

5 Parameters

1: NPDE – INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: TS – ***double precision*** *Input/Output*

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

Constraint: TS $<$ TOUT.

On exit: the value of t corresponding to the solution values in U. Normally TS $=$ TOUT.

3: TOUT – ***double precision*** *Input*

On entry: the final value of t to which the integration is to be carried out.

4: PDEDEF – SUBROUTINE, supplied by the user. *External Procedure*

PDEDEF must evaluate the functions G_i which define the system of PDEs. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PRF.

Its specification is:

```
      SUBROUTINE PDEDEF (NPDE, T, X, U, UDOT, UX, NCODE, V, VDOT, RES,
     1                   IRES)

      INTEGER            NPDE, NCODE, IRES
      double precision   T, X, U(NPDE), UDOT(NPDE), UX(NPDE), V(*),
     1                   VDOT(*), RES(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – ***double precision*** *Input*

On entry: the current value of the independent variable t.

3: X – ***double precision*** *Input*

On entry: the current value of the space variable x.

4: U(NPDE) – ***double precision*** array *Input*

On entry: U(i) contains the value of the component $U_i(x,t)$, for $i = 1, 2, \ldots,$ NPDE.

5: UDOT(NPDE) – ***double precision*** array *Input*

On entry: UDOT(i) contains the value of the component $\frac{\partial U_i(x,t)}{\partial t}$, for $i = 1, 2, \ldots,$ NPDE.

6: UX(NPDE) – ***double precision*** array *Input*

On entry: UX(i) contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$, for $i = 1, 2, \ldots,$ NPDE.

7: NCODE – INTEGER *Input*

On entry: the number of coupled ODEs in the system.

8: V($*$) – ***double precision*** array *Input*

Note: the dimension of the array V must be at least NCODE.

On entry: V(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$NCODE.

9: VDOT($*$) – ***double precision*** array *Input*

Note: the dimension of the array VDOT must be at least NCODE.

On entry: VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$NCODE.

10: RES(NPDE) – ***double precision*** array *Output*

On exit: RES(i) must contain the ith component of G, for $i = 1, 2, \ldots,$NPDE, where G is defined as

$$G_i = \sum_{j=1}^{\text{NPDE}} P_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} Q_{i,j} \dot{V}_j, \tag{9}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i = \sum_{j=1}^{\text{NPDE}} P_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} Q_{i,j} \dot{V}_j + R_i, \tag{10}$$

i.e., all terms in equation (3).

The definition of G is determined by the input value of IRES.

11: IRES – INTEGER *Input/Output*

On entry: the form of G_i that must be returned in the array RES. If IRES $= -1$, then equation (9) must be used. If IRES $= 1$, then equation (10) must be used.

On exit: should usually remain unchanged. However, you may set IRES to force the integration routine to take certain actions, as described below:

IRES $= 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL $= 6$.

IRES $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set IRES $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set IRES $= 3$, then D03PRF returns to the calling (sub)program with the error indicator set to IFAIL $= 4$.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: BNDARY – SUBROUTINE, supplied by the user. *External Procedure*

BNDARY must evaluate the functions G_i^L and G_i^R which describe the boundary conditions, as given in (5) and (6).

Its specification is:

```
      SUBROUTINE BNDARY (NPDE, T, IBND, NOBC, U, UDOT, NCODE, V, VDOT,
     1                   RES, IRES)
      INTEGER            NPDE, IBND, NOBC, NCODE, IRES
      double precision   T, U(NPDE), UDOT(NPDE), V(*), VDOT(*), RES(NOBC)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – ***double precision*** *Input*

On entry: the current value of the independent variable t.

3: IBND – INTEGER *Input*

On entry: specifies which boundary conditions are to be evaluated.

IBND $= 0$

BNDARY must compute the left-hand boundary condition at $x = a$.

IBND $\neq 0$

BNDARY must compute of the right-hand boundary condition at $x = b$.

4: NOBC – INTEGER *Input*

On entry: specifies the number n_a of boundary conditions at the boundary specified by IBND.

5: U(NPDE) – ***double precision*** array *Input*

On entry: U(i) contains the value of the component $U_i(x,t)$ at the boundary specified by IBND, for $i = 1, 2, \ldots,$ NPDE.

6: UDOT(NPDE) – ***double precision*** array *Input*

On entry: UDOT(i) contains the value of the component $\frac{\partial U_i(x,t)}{\partial t}$, for $i = 1, 2, \ldots,$ NPDE.

7: NCODE – INTEGER *Input*

On entry: the number of coupled ODEs in the system.

8: V$(*)$ – ***double precision*** array *Input*

Note: the dimension of the array V must be at least NCODE.

On entry: V(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

9: VDOT$(*)$ – ***double precision*** array *Input*

Note: the dimension of the array VDOT must be at least NCODE.

On entry: VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

Note: VDOT(i), for $i = 1, 2, \ldots,$ NCODE, may only appear linearly as in (11) and (12).

10: RES(NOBC) – ***double precision*** array *Output*

On exit: RES(i) must contain the ith component of G^L or G^R, depending on the value of IBND, for $i = 1, 2, \ldots,$ NOBC, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\text{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} H_{i,j}^L \dot{V}_j, \tag{11}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i^L = \sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L, \tag{12}$$

i.e., all terms in equation (7), and similarly for G_i^R.

The definitions of G^L and G^R are determined by the input value of IRES.

11: IRES – INTEGER *Input/Output*

On entry: the form of G_i^L (or G_i^R) that must be returned in the array RES. If IRES $= -1$, then equation (11) must be used. If IRES $= 1$, then equation (12) must be used.

On exit: should usually remain unchanged. However, you may set IRES to force the integration routine to take certain actions as described below:

IRES $= 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL $= 6$.

IRES $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set IRES $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set IRES $= 3$, then D03PRF returns to the calling (sub)program with the error indicator set to IFAIL $= 4$.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: UVINIT – SUBROUTINE, supplied by the user. *External Procedure*

UVINIT must supply the initial $(t = t_0)$ values of $U(x,t)$ and $V(t)$ for all values of x in the interval $[a,b]$.

Its specification is:

```
SUBROUTINE UVINIT (NPDE, NPTS, NXI, X, XI, U, NCODE, V)
INTEGER              NPDE, NPTS, NXI, NCODE
double precision     X(NPTS), XI(*), U(NPDE,NPTS), V(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: NPTS – INTEGER *Input*

On entry: the number of mesh points in the interval $[a,b]$.

3: NXI – INTEGER *Input*

On entry: the number of ODE/PDE coupling points.

4: X(NPTS) – ***double precision*** array *Input*

On entry: the current mesh. X(i) contains the value of x_i, for $i = 1,2,\dots,\mathrm{NPTS}$.

5: XI$(*)$ – ***double precision*** array *Input*

Note: the dimension of the array XI must be at least NXI.

On entry: XI(i) contains the ODE/PDE coupling point, ξ_i, for $i = 1,2,\dots,\mathrm{NXI}$.

6: U(NPDE,NPTS) – ***double precision*** array *Output*

On exit: U(i,j) contains the value of the component $U_i(x_j, t_0)$, for $i = 1, 2, \ldots, \text{NPDE}$; $j = 1, 2, \ldots, \text{NPTS}$.

7: NCODE – INTEGER *Input*

On entry: the number of coupled ODEs in the system.

8: V($*$) – ***double precision*** array *Output*

Note: the dimension of the array V must be at least NCODE.

On exit: V(i) must contain the value of component $V_i(t_0)$, for $i = 1, 2, \ldots, \text{NCODE}$.

UVINIT must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) – ***double precision*** array *Input/Output*

On entry: if IND $= 1$, the value of U must be unchanged from the previous call.

On exit: U(NPDE $\times$ $(j-1)+i$) contains the computed solution $U_i(x_j, t)$, for $i = 1, 2, \ldots, \text{NPDE}$; $j = 1, 2, \ldots, \text{NPTS}$, and U(NPTS $\times$ NPDE $+ k$) contains $V_k(t)$, for $k = 1, 2, \ldots, \text{NCODE}$, evaluated at $t = \text{TS}$.

8: NPTS – INTEGER *Input*

On entry: the number of mesh points in the interval $[a, b]$.

Constraint: NPTS ≥ 3.

9: X(NPTS) – ***double precision*** array *Input/Output*

On entry: the initial mesh points in the space direction. X(1) must specify the left-hand boundary, a, and X(NPTS) must specify the right-hand boundary, b.

Constraint: X(1) $<$ X(2) $< \cdots <$ X(NPTS).

On exit: the final values of the mesh points.

10: NLEFT – INTEGER *Input*

On entry: the number n_a of boundary conditions at the left-hand mesh point X(1).

Constraint: $0 \leq$ NLEFT $\leq$ NPDE.

11: NCODE – INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

12: ODEDEF – SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the routines F, which define the system of ODEs, as given in (4). If you wish to compute the solution of a system of PDEs only (i.e., NCODE $= 0$), ODEDEF must be the dummy routine D03PEK. (D03PEK is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
      SUBROUTINE ODEDEF (NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                   UCPT, F, IRES)
      INTEGER            NPDE, NCODE, NXI, IRES
      double precision   T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                   UCPX(NPDE,*), UCPT(NPDE,*), F(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – ***double precision*** *Input*

On entry: the current value of the independent variable t.

3: NCODE – INTEGER *Input*

On entry: the number of coupled ODEs in the system.

4: V($*$) – ***double precision*** array *Input*

Note: the dimension of the array V must be at least NCODE.

On entry: V(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

5: VDOT($*$) – ***double precision*** array *Input*

Note: the dimension of the array VDOT must be at least NCODE.

On entry: VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

6: NXI – INTEGER *Input*

On entry: the number of ODE/PDE coupling points.

7: XI($*$) – ***double precision*** array *Input*

Note: the dimension of the array XI must be at least NXI.

On entry: XI(i) contains the ODE/PDE coupling point, ξ_i, for $i = 1, 2, \ldots,$ NXI.

8: UCP(NPDE,$*$) – ***double precision*** array *Input*

Note: the second dimension of the array UCP must be at least max(1, NXI).

On entry: UCP(i,j) contains the value of $U_i(x,t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NXI.

9: UCPX(NPDE,$*$) – ***double precision*** array *Input*

Note: the second dimension of the array UCPX must be at least max(1, NXI).

On entry: UCPX(i,j) contains the value of $\frac{\partial U_i(x,t)}{\partial x}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NXI.

10: UCPT(NPDE,$*$) – ***double precision*** array *Input*

Note: the second dimension of the array UCPT must be at least max(1, NXI).

On entry: UCPT(i,j) contains the value of $\frac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NXI.

11: F($*$) – ***double precision*** array *Output*

Note: the dimension of the array F must be at least NCODE.

On exit: F(i) must contain the ith component of F, for $i = 1, 2, \ldots,$ NCODE, where F is defined as

$$F = -B\dot{V} - CU_t^*, \tag{13}$$

that is, only terms depending explicitly on time derivatives, or

$$F = A - B\dot{V} - CU_t^*, \tag{14}$$

that is, all terms in equation (4). The definition of F is determined by the input value of IRES.

12: IRES – INTEGER *Input/Output*

On entry: the form of F that must be returned in the array F. If IRES $= -1$, then equation (13) must be used. If IRES $= 1$, then equation (14) must be used.

On exit: should usually remain unchanged. However, you may reset IRES to force the integration routine to take certain actions, as described below:

IRES $= 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to IFAIL $= 6$.

IRES $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set IRES $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set IRES $= 3$, then D03PRF returns to the calling (sub)program with the error indicator set to IFAIL $= 4$.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: NXI – INTEGER *Input*

On entry: the number of ODE/PDE coupling points.

Constraints:

if NCODE $= 0$, NXI $= 0$;
if NCODE > 0, NXI ≥ 0.

14: XI($*$) – ***double precision*** array *Input*

Note: the dimension of the array XI must be at least max(1, NXI).

On entry: XI(i), for $i = 1, 2, \ldots,$ NXI, must be set to the ODE/PDE coupling points, ξ_i.

Constraint: X(1) $\leq$ XI(1) $<$ XI(2) $< \cdots <$ XI(NXI) $\leq$ X(NPTS).

15: NEQN – INTEGER *Input*

On entry: the number of ODEs in the time direction.

Constraint: NEQN $=$ NPDE $\times$ NPTS $+$ NCODE.

16: RTOL($*$) – ***double precision*** array *Input*

Note: the dimension of the array RTOL must be at least 1 if ITOL = 1 or 2 and at least NEQN if ITOL = 3 or 4.

On entry: the relative local error tolerance.

Constraint: RTOL(i) ≥ 0 for all relevant i.

17: ATOL($*$) – ***double precision*** array *Input*

Note: the dimension of the array ATOL must be at least 1 if ITOL = 1 or 3 and at least NEQN if ITOL = 2 or 4.

On entry: the absolute local error tolerance.

Constraint: ATOL(i) ≥ 0 for all relevant i.

Note: corresponding elements of RTOL and ATOL cannot both be 0.0.

18: ITOL – INTEGER *Input*

A value to indicate the form of the local error test. ITOL indicates to D03PRF whether to interpret either or both of RTOL or ATOL as a vector or scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

On entry:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1) $\times$ $\lvert$U(i)$\rvert$ + ATOL(1)
2	scalar	vector	RTOL(1) $\times$ $\lvert$U(i)$\rvert$ + ATOL(i)
3	vector	scalar	RTOL(i) $\times$ $\lvert$U(i)$\rvert$ + ATOL(1)
4	vector	vector	RTOL(i) $\times$ $\lvert$U(i)$\rvert$ + ATOL(i)

In the above, e_i denotes the estimated local error for the ith component of the coupled PDE/ODE system in time, U(i), for $i = 1, 2, \ldots,$ NEQN.

The choice of norm used is defined by the parameter NORM.

Constraint: $1 \leq$ ITOL ≤ 4.

19: NORM – CHARACTER*1 *Input*

On entry: the type of norm to be used.

NORM = 'M'

Maximum norm.

NORM = 'A'

Averaged L_2 norm.

If U_{norm} denotes the norm of the vector U of length NEQN, then for the averaged L_2 norm

$$U_{\text{norm}} = \sqrt{\frac{1}{\text{NEQN}} \sum_{i=1}^{\text{NEQN}} (U(i)/w_i)^2},$$

while for the maximum norm

$$U_{\text{norm}} = \max_i |\text{U}(i)/w_i|.$$

See the description of ITOL for the formulation of the weight vector w.

Constraint: NORM = 'M' or 'A'.

20: LAOPT – CHARACTER*1 *Input*

On entry: the type of matrix algebra required.

LAOPT = 'F'

Full matrix methods to be used.

LAOPT = 'B'

Banded matrix methods to be used.

LAOPT = 'S'

Sparse matrix methods to be used.

Constraint: LAOPT = 'F', 'B' or 'S'

Note: you are recommended to use the banded option when no coupled ODEs are present (i.e., NCODE = 0).

21: ALGOPT(30) – ***double precision*** array *Input*

On entry: may be set to control various options available in the integrator. If you wish to employ all the default options, then ALGOPT(1) should be set to 0.0. Default values will also be used for any other elements of ALGOPT set to zero. The permissible values, default values, and meanings are as follows:

ALGOPT(1)

Selects the ODE integration method to be used. If ALGOPT(1) = 1.0, a BDF method is used and if ALGOPT(1) = 2.0, a Theta method is used. The default value is ALGOPT(1) = 1.0.

If ALGOPT(1) = 2.0, then ALGOPT(i), for $i = 2, 3, 4$ are not used.

ALGOPT(2)

Specifies the maximum order of the BDF integration formula to be used. ALGOPT(2) may be 1.0, 2.0, 3.0, 4.0 or 5.0. The default value is ALGOPT(2) = 5.0.

ALGOPT(3)

Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the BDF method. If ALGOPT(3) = 1.0 a modified Newton iteration is used and if ALGOPT(3) = 2.0 a functional iteration method is used. If functional iteration is selected and the integrator encounters difficulty, then there is an automatic switch to the modified Newton iteration. The default value is ALGOPT(3) = 1.0.

ALGOPT(4)

Specifies whether or not the Petzold error test is to be employed. The Petzold error test results in extra overhead but is more suitable when algebraic equations are present, such as $P_{i,j} = 0.0$, for $j = 1, 2, \ldots, \mathrm{NPDE}$ for some i or when there is no $\dot{V}_i(t)$ dependence in the coupled ODE system. If ALGOPT(4) = 1.0, then the Petzold test is used. If ALGOPT(4) = 2.0, then the Petzold test is not used. The default value is ALGOPT(4) = 1.0.

If ALGOPT(1) = 1.0, then ALGOPT(i), for $i = 5, 6, 7$ are not used.

ALGOPT(5)

Specifies the value of Theta to be used in the Theta integration method. $0.51 \le \mathrm{ALGOPT}(5) \le 0.99$. The default value is ALGOPT(5) = 0.55.

ALGOPT(6)

Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the Theta method. If ALGOPT(6) = 1.0, a modified Newton iteration is used and if ALGOPT(6) = 2.0, a functional iteration method is used. The default value is ALGOPT(6) = 1.0.

ALGOPT(7)

Specifies whether or not the integrator is allowed to switch automatically between modified Newton and functional iteration methods in order to be more efficient. If ALGOPT(7) = 1.0, then switching is allowed and if ALGOPT(7) = 2.0, then switching is not allowed. The default value is ALGOPT(7) = 1.0.

ALGOPT(11)

Specifies a point in the time direction, t_{crit}, beyond which integration must not be attempted. The use of t_{crit} is described under the parameter ITASK. If ALGOPT(1) $\neq$ 0.0, a value of 0.0 for ALGOPT(11), say, should be specified even if ITASK subsequently specifies that t_{crit} will not be used.

ALGOPT(12)

Specifies the minimum absolute step size to be allowed in the time integration. If this option is not required, ALGOPT(12) should be set to 0.0.

ALGOPT(13)

Specifies the maximum absolute step size to be allowed in the time integration. If this option is not required, ALGOPT(13) should be set to 0.0.

ALGOPT(14)

Specifies the initial step size to be attempted by the integrator. If ALGOPT(14) = 0.0, then the initial step size is calculated internally.

ALGOPT(15)

Specifies the maximum number of steps to be attempted by the integrator in any one call. If ALGOPT(15) = 0.0, then no limit is imposed.

ALGOPT(23)

Specifies what method is to be used to solve the nonlinear equations at the initial point to initialize the values of U, U_t, V and $\dot{V}$. If ALGOPT(23) = 1.0, a modified Newton iteration is used and if ALGOPT(23) = 2.0, functional iteration is used. The default value is ALGOPT(23) = 1.0.

ALGOPT(29) and ALGOPT(30) are used only for the sparse matrix algebra option, i.e., LAOPT = 'S'.

ALGOPT(29)

Governs the choice of pivots during the decomposition of the first Jacobian matrix. It should lie in the range 0.0 < ALGOPT(29) < 1.0, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If ALGOPT(29) lies outside this range then the default value is used. If the routines regard the Jacobian matrix as numerically singular then increasing ALGOPT(29) towards 1.0 may help, but at the cost of increased fill-in. The default value is ALGOPT(29) = 0.1.

ALGOPT(30)

Used as a relative pivot threshold during subsequent Jacobian decompositions (see ALGOPT(29)) below which an internal error is invoked. ALGOPT(30) must be greater than zero, otherwise the default value is used. If ALGOPT(30) is greater than 1.0 no check is made on the pivot size, and this may be a necessary option if the Jacobian is found to be numerically singular (see ALGOPT(29)). The default value is ALGOPT(30) = 0.0001.

22: REMESH – LOGICAL *Input*

On entry: indicates whether or not spatial remeshing should be performed.

REMESH = .TRUE.

Indicates that spatial remeshing should be performed as specified.

REMESH = .FALSE.

Indicates that spatial remeshing should be suppressed.

Note: REMESH should **not** be changed between consecutive calls to D03PRF. Remeshing can be switched off or on at specified times by using appropriate values for the parameters NRMESH and TRMESH at each call.

23: NXFIX – INTEGER *Input*

On entry: the number of fixed mesh points.

Constraint: $0 \leq \text{NXFIX} \leq \text{NPTS} - 2$

Note: the end points X(1) and X(NPTS) are fixed automatically and hence should not be specified as fixed points.

24: XFIX($*$) – ***double precision*** array *Input*

Note: the dimension of the array XFIX must be at least $\max(1, \text{NXFIX})$.

On entry: XFIX(i), for $i = 1, 2, \ldots, \text{NXFIX}$, must contain the value of the x co-ordinate at the ith fixed mesh point.

Constraint: $\text{XFIX}(i) < \text{XFIX}(i+1)$, for $i = 1, 2, \ldots, \text{NXFIX} - 1$, and each fixed mesh point must coincide with a user-supplied initial mesh point, that is $\text{XFIX}(i) = \text{X}(j)$ for some j, $2 \leq j \leq \text{NPTS} - 1$.

Note: the positions of the fixed mesh points in the array X remain fixed during remeshing, and so the number of mesh points between adjacent fixed points (or between fixed points and end points) does not change. You should take this into account when choosing the initial mesh distribution.

25: NRMESH – INTEGER *Input*

On entry: indicates the form of meshing to be performed.

NRMESH < 0

Indicates that a new mesh is adopted according to the parameter DXMESH. The mesh is tested every |NRMESH| timesteps.

NRMESH $= 0$

Indicates that remeshing should take place just once at the end of the first time step reached when $t >$ TRMESH.

NRMESH > 0

Indicates that remeshing will take place every NRMESH time steps, with no testing using DXMESH.

Note: NRMESH may be changed between consecutive calls to D03PRF to give greater flexibility over the times of remeshing.

26: DXMESH – ***double precision*** *Input*

On entry: determines whether a new mesh is adopted when NRMESH is set less than zero. A possible new mesh is calculated at the end of every |NRMESH| time steps, but is adopted only if

$$x_i^{\text{new}} > x_i^{\text{old}} + \text{DXMESH} \times \left(x_{i+1}^{\text{old}} - x_i^{\text{old}}\right),$$

or

$$x_i^{\text{new}} < x_i^{\text{old}} - \text{DXMESH} \times \left(x_i^{\text{old}} - x_{i-1}^{\text{old}}\right).$$

DXMESH thus imposes a lower limit on the difference between one mesh and the next.

Constraint: DXMESH ≥ 0.0.

27: TRMESH – ***double precision*** *Input*

On entry: specifies when remeshing will take place when NRMESH is set to zero. Remeshing will occur just once at the end of the first time step reached when t is greater than TRMESH.

Note: TRMESH may be changed between consecutive calls to D03PRF to force remeshing at several specified times.

28: IPMINF – INTEGER *Input*

On entry: the level of trace information regarding the adaptive remeshing. Details are directed to the current advisory message unit (see X04ABF).

IPMINF = 0

No trace information.

IPMINF = 1

Brief summary of mesh characteristics.

IPMINF = 2

More detailed information, including old and new mesh points, mesh sizes and monitor function values.

Constraint: $0 \leq$ IPMINF ≤ 2.

29: XRATIO – ***double precision*** *Input*

On entry: input bound on adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The remeshing routines will attempt to ensure that

$$(x_i - x_{i-1})/\mathrm{XRATIO} < x_{i+1} - x_i < \mathrm{XRATIO} \times (x_i - x_{i-1}).$$

Suggested value: XRATIO = 1.5.

Constraint: XRATIO > 1.0.

30: CON – ***double precision*** *Input*

On entry: an input bound on the sub-integral of the monitor function $F^{\mathrm{mon}}(x)$ over each space step. The remeshing routines will attempt to ensure that

$$\int_{x_1}^{x_{i+1}} F^{\mathrm{mon}}(x)\, dx \leq \mathrm{CON} \int_{x_1}^{x_{\mathrm{NPTS}}} F^{\mathrm{mon}}(x)\, dx,$$

(see Furzeland (1984)). CON gives you more control over the mesh distribution e.g., decreasing CON allows more clustering. A typical value is $2/(\mathrm{NPTS} - 1)$, but you are encouraged to experiment with different values. Its value is not critical and the mesh should be qualitatively correct for all values in the range given below.

Suggested value: CON $= 2.0/(\mathrm{NPTS} - 1)$.

Constraint: $0.1/(\mathrm{NPTS} - 1) \leq \mathrm{CON} \leq 10.0/(\mathrm{NPTS} - 1)$.

31: MONITF – SUBROUTINE, supplied by the user. *External Procedure*

MONITF must supply and evaluate a remesh monitor function to indicate the solution behaviour of interest.

If you specify REMESH = .FALSE., i.e., no remeshing, then MONITF will not be called and the dummy routine D03PEL may be used for MONITF. (D03PEL is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
SUBROUTINE MONITF (T, NPTS, NPDE, X, U, FMON)
INTEGER              NPTS, NPDE
```
double precision `T, X(NPTS), U(NPDE,NPTS), FMON(NPTS)`

1: T – ***double precision*** *Input*

On entry: the current value of the independent variable t.

2: NPTS – INTEGER *Input*

On entry: the number of mesh points in the interval $[a, b]$.

3: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

4: X(NPTS) – ***double precision*** array *Input*

On entry: the current mesh. X(i) contains the value of x_i, for $i = 1, 2, \ldots,$ NPTS.

5: U(NPDE,NPTS) – ***double precision*** array *Input*

Note: the second dimension of the array U must be at least NPDE × NPTS.

On entry: U(i,j) contains the value of $U_i(x, t)$ at $x =$ X(j) and time t, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NPTS.

6: FMON(NPTS) – ***double precision*** array *Output*

On exit: FMON(i) must contain the value of the monitor function $F^{\mathrm{mon}}(x)$ at mesh point $x =$ X(i).

MONITF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

32: RSAVE(LRSAVE) – ***double precision*** array *Communication Array*

If IND = 0, RSAVE need not be set on entry.

If IND = 1, RSAVE must be unchanged from the previous call to the routine because it contains required information about the iteration.

33: LRSAVE – INTEGER *Input*

On entry: the dimension of the array RSAVE as declared in the (sub)program from which D03PRF is called. Its size depends on the type of matrix algebra selected.

If LAOPT = 'F', LRSAVE $\geq$ NEQN × NEQN + NEQN + $NWKRES + LENODE$.

If LAOPT = 'B', LRSAVE $\geq$ (3 × $ML + MU$ + 2) × NEQN + $NWKRES + LENODE$.

If LAOPT = 'S', LRSAVE $\geq$ 4 × NEQN + 11 × NEQN/2 + 1 + $NWKRES + LENODE$.

Where

ML and MU are the lower and upper half bandwidths given by
ML = NPDE + NLEFT − 1, and
MU = 2 × NPDE − NLEFT − 1, for problems involving PDEs only, and
$ML = MU$ = NEQN − 1, for coupled PDE/ODE problems.

$NWKRES$ = NPDE × (3 × NPDE + 6 × NXI + NPTS + 15) + NXI + NCODE + 7 × NPTS + NXFIX + 1, when NCODE > 0 and NXI > 0, and
$NWKRES$ = NPDE × (3 × NPDE + NPTS + 21) + NCODE + 7 × NPTS + NXFIX + 2,
when NCODE > 0 and NXI = 0, and

$NWKRES = \text{NPDE} \times (3 \times \text{NPDE} + \text{NPTS} + 21) + 7 \times \text{NPTS} + \text{NXFIX} + 3$, when NCODE = 0.

$LENODE = (6 + \text{int}(\text{ALGOPT}(2))) \times \text{NEQN} + 50$, when the BDF method is used, and
$LENODE = 9 \times \text{NEQN} + 50$, when the Theta method is used.

Note: when using the sparse option, the value of LRSAVE may be too small when supplied to the integrator. An estimate of the minimum size of LRSAVE is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

34: ISAVE(LISAVE) – INTEGER array *Communication Array*

If IND = 0, ISAVE need not be set.

If IND = 1, ISAVE must be unchanged from the previous call to the routine because it contains required information about the iteration. In particular the following components of the array ISAVE concern the efficiency of the integration:

ISAVE(1)

Contains the number of steps taken in time.

ISAVE(2)

Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves evaluating the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

ISAVE(3)

Contains the number of Jacobian evaluations performed by the time integrator.

ISAVE(4)

Contains the order of the ODE method last used in the time integration.

ISAVE(5)

Contains the number of Newton iterations performed by the time integrator. Each iteration involves residual evaluation of the resulting ODE system followed by a back-substitution using the *LU* decomposition of the Jacobian matrix.

The rest of the array is used as workspace.

35: LISAVE – INTEGER *Input*

On entry: the dimension of the array ISAVE as declared in the (sub)program from which D03PRF is called. Its size depends on the type of matrix algebra selected:

if LAOPT = 'F', LISAVE $\geq 25 + \text{NXFIX}$;
if LAOPT = 'B', LISAVE $\geq \text{NEQN} + 25 + \text{NXFIX}$;
if LAOPT = 'S', LISAVE $\geq 25 \times \text{NEQN} + 25 + \text{NXFIX}$.

Note: when using the sparse option, the value of LISAVE may be too small when supplied to the integrator. An estimate of the minimum size of LISAVE is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

36: ITASK – INTEGER *Input*

On entry: the task to be performed by the ODE integrator.

ITASK = 1

Normal computation of output values U at t = TOUT (by overshooting and interpolating).

ITASK = 2

Take one step in the time direction and return.

ITASK = 3

Stop at first internal integration point at or beyond $t = \text{TOUT}$.

ITASK = 4

Normal computation of output values U at $t = \text{TOUT}$ but without overshooting $t = t_{\text{crit}}$, where t_{crit} is described under the parameter ALGOPT.

ITASK = 5

Take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: $1 \leq \text{ITASK} \leq 5$.

37: ITRACE – INTEGER *Input*

On entry: the level of trace information required from D03PRF and the underlying ODE solver as follows:

ITRACE ≤ -1

No output is generated.

ITRACE = 0

Only warning messages from the PDE solver are printed on the current error message unit (see X04AAF).

ITRACE = 1

Output from the underlying ODE solver is printed on the current advisory message unit (see X04ABF). This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

ITRACE = 2

Output from the underlying ODE solver is similar to that produced when ITRACE = 1, except that the advisory messages are given in greater detail.

ITRACE ≥ 3

The output from the underlying ODE solver is similar to that produced when ITRACE = 2, except that the advisory messages are given in greater detail.

You are advised to set ITRACE = 0, unless you are experienced with sub-chapter D02M/N.

38: IND – INTEGER *Input/Output*

On entry: must be set to 0 or 1.

IND = 0

Starts or restarts the integration in time.

IND = 1

Continues the integration after an earlier exit from the routine. In this case, only the parameters TOUT and IFAIL and the remeshing parameters NRMESH, DXMESH, TRMESH, XRATIO and CON may be reset between calls to D03PRF.

Constraint: $0 \leq \text{IND} \leq 1$.

On exit: IND = 1.

39: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

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, if you are not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

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 $= 1$

On entry, (TOUT – TS) is too small,
or ITASK $\neq$ 1, 2, 3, 4 or 5,
or at least one of the coupling points defined in array XI is outside the interval $[\mathrm{X}(1), \mathrm{X}(\mathrm{NPTS})]$,
or NPTS < 3,
or NPDE < 1,
or NLEFT not in the range 0 to NPDE,
or NORM $\neq$ 'A' or 'M',
or LAOPT $\neq$ 'F', 'B' or 'S',
or ITOL $\neq$ 1, 2, 3 or 4,
or IND $\neq$ 0 or 1,
or mesh points X(i) badly ordered,
or LRSAVE is too small,
or LISAVE is too small,
or NCODE and NXI are incorrectly defined,
or IND $= 1$ on initial entry to D03PRF,
or an element of RTOL or ATOL < 0.0,
or corresponding elements of RTOL and ATOL are both 0.0,
or NEQN $\neq$ NPDE $\times$ NPTS $+$ NCODE,
or NXFIX not in the range 0 to NPTS – 2,
or fixed mesh point(s) do not coincide with any of the user-supplied mesh points,
or DXMESH < 0.0,
or IPMINF $\neq$ 0, 1 or 2,
or XRATIO ≤ 1.0,
or CON not in the range 0.1/(NPTS – 1) to 10/(NPTS – 1).

IFAIL $= 2$

The underlying ODE solver cannot make any further progress, with the values of ATOL and RTOL, across the integration range from the current point $t =$ TS. The components of U contain the computed values at the current point $t =$ TS.

IFAIL $= 3$

In the underlying ODE solver, there were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as $t =$ TS. The problem may have a singularity, or the error requirement may be inappropriate. Incorrect positioning of boundary conditions may also result in this error.

IFAIL $= 4$

In setting up the ODE system, the internal initialization routine was unable to initialize the derivative of the ODE system. This could be due to the fact that IRES was repeatedly set to 3 in one of the user-supplied (sub)programs PDEDEF, BNDARY or ODEDEF, when the residual in the underlying ODE solver was being evaluated. Incorrect positioning of boundary conditions may also result in this error.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. You should check their problem formulation.

IFAIL = 6

When evaluating the residual in solving the ODE system, IRES was set to 2 in one of the user-supplied (sub)programs PDEDEF, BNDARY or ODEDEF. Integration was successful as far as $t =$ TS.

IFAIL = 7

The values of ATOL and RTOL are so small that the routine is unable to start the integration in time.

IFAIL = 8

In one of the user-supplied (sub)programs, PDEDEF, BNDARY or ODEDEF, IRES was set to an invalid value.

IFAIL = 9 (D02NNF)

A serious error has occurred in an internal call to the specified routine. Check problem specification an all parameters and array dimensions. Setting ITRACE = 1 may provide more information. If the problem persists, contact NAG.

IFAIL = 10

The required task has been completed, but it is estimated that a small change in ATOL and RTOL is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when ITASK $\neq$ 2 or 5.)

IFAIL = 11

An error occurred during Jacobian formulation of the ODE system (a more detailed error description may be directed to the current advisory message unit). If using the sparse matrix algebra option, the values of ALGOPT(29) and ALGOPT(30) may be inappropriate.

IFAIL = 12

In solving the ODE system, the maximum number of steps specified in ALGOPT(15) have been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see the description of ITOL). Pure relative error control (ATOL(i) = 0.0) was requested on a variable (the ith) which has become zero. The integration was successful as far as $t =$ TS.

IFAIL = 14

Not applicable.

IFAIL = 15

When using the sparse option, the value of LISAVE or LRSAVE was insufficient (more detailed information may be directed to the current error message unit).

IFAIL = 16

REMESH has been changed between calls to D03PRF.

7 Accuracy

D03PRF controls the accuracy of the integration in the time direction but not the accuracy of the approximation in space. The spatial accuracy depends on both the number of mesh points and on their distribution in space. In the time integration only the local error over a single step is controlled and so the accuracy over a number of steps cannot be guaranteed. You should therefore test the effect of varying the accuracy parameters, ATOL and RTOL.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first-order by the introduction of new variables (see the example in Section 9). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (D03PPF/D03PPA for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretization scheme with upwind weighting (D03PSF for example), or the addition of a second-order artificial dissipation term.

The time taken depends on the complexity of the system, the accuracy requested, and the frequency of the mesh updates. For a given system with fixed accuracy and mesh-update frequency it is approximately proportional to NEQN.

9 Example

This example is the first-order system

$$\begin{aligned}\frac{\partial U_1}{\partial t}+\frac{\partial U_1}{\partial x}+\frac{\partial U_2}{\partial x} &= 0,\\ \frac{\partial U_2}{\partial t}+4\frac{\partial U_1}{\partial x}+\frac{\partial U_2}{\partial x} &= 0,\end{aligned}$$

for $x \in [0,1]$ and $t \geq 0$.

The initial conditions are

$$\begin{aligned}U_1(x,0) &= e^x,\\ U_2(x,0) &= x^2+\sin(2\pi x^2),\end{aligned}$$

and the Dirichlet boundary conditions for U_1 at $x=0$ and U_2 at $x=1$ are given by the exact solution:

$$\begin{aligned}U_1(x,t) &= \tfrac{1}{2}\{e^{x+t}+e^{x-3t}\}+\tfrac{1}{4}\Big\{\sin\Big(2\pi(x-3t)^2\Big)-\sin\Big(2\pi(x+t)^2\Big)\Big\}+2t^2-2xt,\\ U_2(x,t) &= e^{x-3t}-e^{x+t}+\tfrac{1}{2}\Big\{\sin\Big(2\pi(x-3t)^2\Big)+\sin\Big(2\pi(x+t)^2\Big)\Big\}+x^2+5t^2-2xt.\end{aligned}$$

9.1 Program Text

```
*     D03PRF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NV, NXI, NXFIX, NLEFT, NEQN, NIW,
     +                 NWKRES, LENODE, NW, INTPTS, ITYPE
      PARAMETER        (NPDE=2,NPTS=61,NV=0,NXI=0,NXFIX=0,NLEFT=1,
     +                 NEQN=NPDE*NPTS+NV,NIW=25+NXFIX,
     +                 NWKRES=NPDE*(NPTS+21+3*NPDE)+7*NPTS+NXFIX+3,
     +                 LENODE=11*NEQN+50,NW=NEQN*NEQN+NEQN+NWKRES+
     +                 LENODE,INTPTS=5,ITYPE=1)
*     .. Scalars in Common ..
      DOUBLE PRECISION P
```

```
*     .. Local Scalars ..
      DOUBLE PRECISION CONST, DXMESH, TOUT, TRMESH, TS, XRATIO, XX
      INTEGER          I, IFAIL, IND, IPMINF, IT, ITASK, ITOL, ITRACE,
     +                 NRMESH
      LOGICAL          REMESH, THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      DOUBLE PRECISION ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UE(NPDE,NPTS), UOUT(NPDE,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XFIX(1), XI(1), XOUT(INTPTS)
      INTEGER          IW(NIW)
*     .. External Functions ..
      DOUBLE PRECISION X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PEK, D03PRF, D03PZF, EXACT, MONITF,
     +                 PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PRF Example Program Results'
      P = X01AAF(XX)
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 0.5D-4
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99996) ATOL, NPTS
*
*     Set remesh parameters ..
*
      REMESH = .TRUE.
      NRMESH = 3
      DXMESH = 0.0D0
      CONST = 5.0D0/(NPTS-1.0D0)
      XRATIO = 1.2D0
      IPMINF = 0
      WRITE (NOUT,99999) NRMESH
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0D0)/(NPTS-1.0D0)
   20 CONTINUE
*
      XOUT(1) = 0.0D0
      XOUT(2) = 0.25D0
      XOUT(3) = 0.5D0
      XOUT(4) = 0.75D0
      XOUT(5) = 1.0D0
      WRITE (NOUT,99998) (XOUT(I),I=1,INTPTS)
*
      XI(1) = 0.0D0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0D0
   40 CONTINUE
      IF (THETA) THEN
         ALGOPT(1) = 2.0D0
         ALGOPT(6) = 2.0D0
         ALGOPT(7) = 1.0D0
      END IF
*
*     Loop over output value of t
*
```

```
      TS = 0.0D0
      TOUT = 0.0D0
*
      DO 60 IT = 1, 5
         TOUT = 0.05D0*IT
         IFAIL = 0
*
         CALL D03PRF(NPDE,TS,TOUT,PDEDEF,BNDARY,UVINIT,U,NPTS,X,NLEFT,
     +               NV,D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,REMESH,NXFIX,XFIX,NRMESH,DXMESH,TRMESH,
     +               IPMINF,XRATIO,CONST,MONITF,W,NW,IW,NIW,ITASK,
     +               ITRACE,IND,IFAIL)
*
*        Interpolate at output points ..
         CALL D03PZF(NPDE,0,U,NPTS,X,XOUT,INTPTS,ITYPE,UOUT,IFAIL)
*        Check against exact solution ..
         CALL EXACT(TS,NPDE,INTPTS,XOUT,UE)
*
         WRITE (NOUT,99997) TS
         WRITE (NOUT,99994) (UOUT(1,I,1),I=1,INTPTS)
         WRITE (NOUT,99993) (UE(1,I),I=1,INTPTS)
         WRITE (NOUT,99992) (UOUT(2,I,1),I=1,INTPTS)
         WRITE (NOUT,99991) (UE(2,I),I=1,INTPTS)
*
   60 CONTINUE
      WRITE (NOUT,99995) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' Remeshing every ',I3,' time steps',/)
99998 FORMAT (' X        ',5F10.4,/)
99997 FORMAT (' T = ',F6.3)
99996 FORMAT (//'  Accuracy requirement =',E10.3,' Number of points = ',
     +       I3,/)
99995 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6)
99994 FORMAT (' Approx U1',5F10.4)
99993 FORMAT (' Exact  U1',5F10.4)
99992 FORMAT (' Approx U2',5F10.4)
99991 FORMAT (' Exact  U2',5F10.4,/)
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,NXI,X,XI,U,NV,V)
*     .. Scalar Arguments ..
      INTEGER           NPDE, NPTS, NV, NXI
*     .. Array Arguments ..
      DOUBLE PRECISION  U(NPDE,NPTS), V(*), X(NPTS), XI(*)
*     .. Scalars in Common ..
      DOUBLE PRECISION  P
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP, SIN
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(1,I) = EXP(X(I))
         U(2,I) = X(I)**2 + SIN(2.0D0*P*X(I)**2)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UDOT,DUDX,NV,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      DOUBLE PRECISION  T, X
      INTEGER           IRES, NPDE, NV
*     .. Array Arguments ..
      DOUBLE PRECISION  DUDX(NPDE), RES(NPDE), U(NPDE), UDOT(NPDE),
     +                  V(*), VDOT(*)
*     .. Executable Statements ..
```

```
      IF (IRES.EQ.-1) THEN
         RES(1) = UDOT(1)
         RES(2) = UDOT(2)
      ELSE
         RES(1) = UDOT(1) + DUDX(1) + DUDX(2)
         RES(2) = UDOT(2) + 4.0D0*DUDX(1) + DUDX(2)
      END IF
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,IBND,NOBC,U,UDOT,NV,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      DOUBLE PRECISION  T
      INTEGER           IBND, IRES, NOBC, NPDE, NV
*     .. Array Arguments ..
      DOUBLE PRECISION  RES(NOBC), U(NPDE), UDOT(NPDE), V(*), VDOT(*)
*     .. Scalars in Common ..
      DOUBLE PRECISION  P
*     .. Local Scalars ..
      DOUBLE PRECISION  PP
*     .. Intrinsic Functions ..
      INTRINSIC         EXP, SIN
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      PP = 2.0D0*P
      IF (IBND.EQ.0) THEN
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0D0
         ELSE
            RES(1) = U(1) - 0.5D0*(EXP(T)+EXP(-3.0D0*T)) -
     +               0.25D0*(SIN(PP*9.0D0*T**2)-SIN(PP*T**2)) -
     +               2.0D0*T**2
         END IF
      ELSE
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0D0
         ELSE
            RES(1) = U(2) - (EXP(1.0D0-3.0D0*T)-EXP(1.0D0+T)
     +               +0.5D0*(SIN(PP*(1.0D0-3.0D0*T)**2)+SIN(PP*(1.0D0+T)
     +               **2))+1.0D0+5.0D0*T**2-2.0D0*T)
         END IF
      END IF
      RETURN
      END
*
      SUBROUTINE EXACT(T,NPDE,NPTS,X,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      DOUBLE PRECISION T
      INTEGER          NPDE, NPTS
*     .. Array Arguments ..
      DOUBLE PRECISION U(NPDE,NPTS), X(NPTS)
*     .. Scalars in Common ..
      DOUBLE PRECISION P
*     .. Local Scalars ..
      DOUBLE PRECISION PP
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        EXP, SIN
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Executable Statements ..
      PP = 2.0D0*P
      DO 20 I = 1, NPTS
         U(1,I) = 0.5D0*(EXP(X(I)+T)+EXP(X(I)-3.0D0*T)) +
     +            0.25D0*(SIN(PP*(X(I)-3.0D0*T)**2)-SIN(PP*(X(I)+T)**2))
     +            + 2.0D0*T**2 - 2.0D0*X(I)*T
         U(2,I) = EXP(X(I)-3.0D0*T) - EXP(X(I)+T) + 0.5D0*(SIN(PP*(X(I)
     +            -3.0D0*T)**2)+SIN(PP*(X(I)+T)**2)) + X(I)**2 +
     +            5.0D0*T**2 - 2.0D0*X(I)*T
```

```
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE MONITF(T,NPTS,NPDE,X,U,FMON)
*     .. Scalar Arguments ..
      DOUBLE PRECISION  T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      DOUBLE PRECISION  FMON(NPTS), U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      DOUBLE PRECISION  D2X1, D2X2, H1, H2, H3
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         ABS, MAX
*     .. Executable Statements ..
      DO 20 I = 2, NPTS - 1
         H1 = X(I) - X(I-1)
         H2 = X(I+1) - X(I)
         H3 = 0.5D0*(X(I+1)-X(I-1))
*        Second derivatives ..
         D2X1 = ABS(((U(1,I+1)-U(1,I))/H2-(U(1,I)-U(1,I-1))/H1)/H3)
         D2X2 = ABS(((U(2,I+1)-U(2,I))/H2-(U(2,I)-U(2,I-1))/H1)/H3)
         FMON(I) = MAX(D2X1,D2X2)
   20 CONTINUE
      FMON(1) = FMON(2)
      FMON(NPTS) = FMON(NPTS-1)
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D03PRF Example Program Results


  Accuracy requirement = 0.500E-04 Number of points =  61

 Remeshing every   3 time steps

 X            0.0000    0.2500    0.5000    0.7500    1.0000

 T =  0.050
 Approx U1    0.9923    1.0894    1.4686    2.3388    2.1071
 Exact  U1    0.9923    1.0893    1.4686    2.3391    2.1073
 Approx U2   -0.0997    0.1057    0.7180    0.0967    0.2021
 Exact  U2   -0.0998    0.1046    0.7193    0.0966    0.2022

 T =  0.100
 Approx U1    1.0613    0.9856    1.3120    2.3084    2.1039
 Exact  U1    1.0613    0.9851    1.3113    2.3092    2.1025
 Approx U2   -0.0150   -0.0481    0.1075   -0.3240    0.3753
 Exact  U2   -0.0150   -0.0495    0.1089   -0.3235    0.3753

 T =  0.150
 Approx U1    1.1485    0.9763    1.2658    2.0906    2.2027
 Exact  U1    1.1485    0.9764    1.2654    2.0911    2.2027
 Approx U2    0.1370   -0.0250   -0.4107   -0.8577    0.3096
 Exact  U2    0.1366   -0.0266   -0.4100   -0.8567    0.3096

 T =  0.200
 Approx U1    1.0956    1.0529    1.3407    1.8322    2.2035
 Exact  U1    1.0956    1.0515    1.3393    1.8327    2.2050
 Approx U2    0.0381    0.1282   -0.7979   -1.1776   -0.4221
 Exact  U2    0.0370    0.1247   -0.7961   -1.1784   -0.4221

 T =  0.250
```

```
Approx U1    0.8119    1.1288    1.5163    1.6076    2.2027
Exact  U1    0.8119    1.1276    1.5142    1.6091    2.2035
Approx U2   -0.4968    0.2123   -1.0259   -1.2149   -1.3938
Exact  U2   -0.4992    0.2078   -1.0257   -1.2183   -1.3938

Number of integration steps in time =     50
Number of function evaluations =   2579
Number of Jacobian evaluations =    20
Number of iterations =    126
```