NAG Toolbox for MATLAB

d03pr

1 Purpose

d03pr 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 Syntax

```
[ts, u, x, rsave, isave, ind, ifail] = d03pr(npde, ts, tout, pdedef,
bndary, uvinit, u, x, nleft, ncode, odedef, xi, rtol, atol, itol,
norm_p, laopt, algopt, remesh, xfix, nrmesh, dxmesh, trmesh, ipminf,
monitf, rsave, isave, itask, itrace, ind, 'npts', npts, 'nxi', nxi,
'neqn', neqn, 'nxfix', nxfix, 'xratio', xratio, 'con', con, 'lrsave',
lrsave, 'lisave', lisave)
```

3 Description

d03pr integrates the system of first-order PDEs and coupled ODEs given by the master equations:

$$G_i\left(x,t,U,U_x,U_t,V,\dot{V}\right)=0,\qquad i=1,2,\ldots,\mathbf{npde},\qquad a\le x\le b, t\ge t_0,\tag{1}$$

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

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

$$G_i=\sum_{j=1}^{\mathbf{npde}}P_{i,j}\frac{\partial U_j}{\partial t}+\sum_{j=1}^{\mathbf{ncode}}Q_{i,j}\dot{V}_j+R_i=0,\qquad i=1,2,\ldots,\mathbf{npde},\tag{3}$$

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

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

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

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

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 $F=\left[F_1,\ldots,F_{\mathbf{ncode}}\right]^{\mathrm{T}}$, A is a vector of length **ncode**, B is an **ncode** by **ncode** matrix, C is an **ncode** by $\left(n_\xi\times\mathbf{npde}\right)$ 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 (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 $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$ 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 user-supplied (sub)program **uvinit**. 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 $n_a + n_b = \mathbf{npde}$. 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 functions.

The boundary conditions have the master equation form:

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

at the left-hand boundary, and

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

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 functions. 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:

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^L \dot{V}_j + S_i^L = 0, \qquad i = 1,2,\ldots,n_a, \tag{7}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^R \dot{V}_j + S_i^R = 0, \qquad i = 1,2,\ldots,n_b, \tag{8}$$

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 user-supplied (sub)program **bndary**.

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 $\mathbf{x}(i)$, for $i = 1,2,\ldots,\mathbf{npts}$, 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 nonzero 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 (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

5.1 Compulsory Input Parameters

1: **npde – int32 scalar**

the number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **ts – double scalar**

The initial value of the independent variable t.

Constraint: **ts** $<$ **tout**.

3: **tout – double scalar**

The final value of t to which the integration is to be carried out.

4: **pdedef – string containing name of m-file**

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 d03pr.

Its specification is:

```
[res, ires] = pdedef(npde, t, x, u, udot, ux, ncode, v, vdot, ires)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **x – double scalar**

The current value of the space variable x.

4: **u(npde) – double array**

u(i) contains the value of the component $U_i(x,t)$, for $i = 1,2,\dots,$**npde**.

5: **udot(npde) – double array**

udot(i) contains the value of the component $\frac{\partial U_i(x,t)}{\partial t}$, for $i = 1,2,\dots,$**npde**.

6: **ux(npde) – double array**

ux(i) contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$, for $i = 1,2,\dots,$**npde**.

7: **ncode – int32 scalar**

The number of coupled ODEs in the system.

8: **v(∗) – double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t)$, for $i = 1,2,\dots,$**ncode**.

9: **vdot(∗) – double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

vdot(i) contains the value of component $\dot{V}_i(t)$, for $i = 1,2,\dots,$**ncode**.

10: **ires – int32 scalar**

The form of G_i that must be returned in the array **res**.

ires $= -1$

Equation (9) must be used.

ires $= 1$

Equation (10) must be used.

Should usually remain unchanged. However, you may set **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **res**(**npde**) – **double array**

res(i) must contain the ith component of G, for $i = 1, 2, \ldots,$ **npde**, where G is defined as

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

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

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

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

The definition of G is determined by the input value of **ires**.

2: **ires** – **int32 scalar**

The form of G_i that must be returned in the array **res**.

ires $= -1$

Equation (9) must be used.

ires $= 1$

Equation (10) must be used.

Should usually remain unchanged. However, you may set **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

5: **bndary – string containing name of m-file**

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:

```
[res, ires] = bndary(npde, t, ibnd, nobc, u, udot, ncode, v, vdot,
ires)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **ibnd – int32 scalar**

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 – int32 scalar**

Specifies the number n_a of boundary conditions at the boundary specified by **ibnd**.

5: **u**(**npde**) **– double array**

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 array**

udot(i) contains the value of the component $\dfrac{\partial U_i(x,t)}{\partial t}$, for $i = 1,2,\ldots,$ **npde**.

7: **ncode – int32 scalar**

The number of coupled ODEs in the system.

8: **v**($*$) **– double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t)$, for $i = 1,2,\ldots,$ **ncode**.

9: **vdot**($*$) **– double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

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: **ires – int32 scalar**

The form of G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$

Equation (11) must be used.

ires $= 1$

Equation (12) must be used.

Should usually remain unchanged. However, you may set **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **res(nobc) – double array**

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

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

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

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

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**.

2: **ires – int32 scalar**

The form of G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$

Equation (11) must be used.

ires $= 1$

Equation (12) must be used.

Should usually remain unchanged. However, you may set **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

6: **uvinit – string containing name of m-file**

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:

```
[u, v] = uvinit(npde, npts, nxi, x, xi, ncode)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **npts – int32 scalar**

The number of mesh points in the interval $[a, b]$.

3: **nxi – int32 scalar**

The number of ODE/PDE coupling points.

4: **x(npts) – double array**

The current mesh. **x**(i) contains the value of x_i, for $i = 1, 2, \ldots,$ **npts**.

5: **xi**$(*)$ **– double array**

Note: the dimension of the array **xi** must be at least **nxi**.

xi(i) contains the ODE/PDE coupling point, ξ_i, for $i = 1, 2, \ldots,$ **nxi**.

6: **ncode – int32 scalar**

The number of coupled ODEs in the system.

Output Parameters

1: **u(npde,npts) – double array**

u(i,j) contains the value of the component $U_i(x_j, t_0)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**.

2: **v**$(*)$ **– double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) must contain the value of component $V_i(t_0)$, for $i = 1, 2, \ldots,$ **ncode**.

7: **u(neqn) – double array**

If **ind** $= 1$, the value of **u** must be unchanged from the previous call.

8: **x(npts) – double array**

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: $\mathbf{x}(1) < \mathbf{x}(2) < \cdots < \mathbf{x}(\mathbf{npts})$.

9: **nleft – int32 scalar**

The number n_a of boundary conditions at the left-hand mesh point **x**(1).

Constraint: $0 \leq \mathbf{nleft} \leq \mathbf{npde}$.

10: **ncode – int32 scalar**

The number of coupled ODE components.

Constraint: $\mathbf{ncode} \geq 0$.

11: **odedef – string containing name of m-file**

odedef must evaluate the functions 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 string `'d03pek'`. **d03pek** is included in the NAG Fortran Library.

Its specification is:

```
[f, ires] = odedef(npde, t, ncode, v, vdot, nxi, xi, ucp, ucpx,
ucpt, ires)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **ncode – int32 scalar**

The number of coupled ODEs in the system.

4: **v(*) – double array**

Note: the dimension of the array **v** must be at least **ncode**.

v(i) contains the value of component $V_i(t)$, for $i = 1, 2, \ldots, \mathbf{ncode}$.

5: **vdot(*) – double array**

Note: the dimension of the array **vdot** must be at least **ncode**.

vdot(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots, \mathbf{ncode}$.

6: **nxi – int32 scalar**

The number of ODE/PDE coupling points.

7: **xi(*) – double array**

Note: the dimension of the array **xi** must be at least **nxi**.

xi(i) contains the ODE/PDE coupling point, ξ_i, for $i = 1, 2, \ldots, \mathbf{nxi}$.

8: **ucp(npde,$*$) – double array**

The first dimension of the array **ucp** must be at least

The second dimension of the array must be at least $\max(1,\mathbf{nxi})$

ucp(i,j) contains the value of $U_i(x,t)$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\mathbf{npde}$ and $j = 1,2,\ldots,\mathbf{nxi}$.

9: **ucpx(npde,$*$) – double array**

The first dimension of the array **ucpx** must be at least

The second dimension of the array must be at least $\max(1,\mathbf{nxi})$

ucpx(i,j) contains the value of $\dfrac{\partial U_i(x,t)}{\partial x}$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\mathbf{npde}$ and $j = 1,2,\ldots,\mathbf{nxi}$.

10: **ucpt(npde,$*$) – double array**

The first dimension of the array **ucpt** must be at least

The second dimension of the array must be at least $\max(1,\mathbf{nxi})$

ucpt(i,j) contains the value of $\dfrac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\mathbf{npde}$ and $j = 1,2,\ldots,\mathbf{nxi}$.

11: **ires – int32 scalar**

The form of **f** that must be returned in the array **f**.

ires $= -1$

Equation (13) must be used.

ires $= 1$

Equation (14) must be used.

Should usually remain unchanged. However, you may reset **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **f($*$) – double array**

Note: the dimension of the array **f** must be at least **ncode**.

f(i) must contain the ith component of **f**, for $i = 1,2,\ldots,\mathbf{ncode}$, 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**.

2: **ires – int32 scalar**

The form of **f** that must be returned in the array **f**.

ires $= -1$

Equation (13) must be used.

ires $= 1$

Equation (14) must be used.

Should usually remain unchanged. However, you may reset **ires** to force the integration function 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 d03pr returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

12: **xi**$(*)$ **– double array**

Note: the dimension of the array **xi** must be at least $\max(1, \mathbf{nxi})$.

xi(i), for $i = 1, 2, \ldots, \mathbf{nxi}$, must be set to the ODE/PDE coupling points, ξ_i.

Constraint: $\mathbf{x}(1) \leq \mathbf{xi}(1) < \mathbf{xi}(2) < \cdots < \mathbf{xi}(\mathbf{nxi}) \leq \mathbf{x}(\mathbf{npts})$.

13: **rtol**$(*)$ **– double array**

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.

The relative local error tolerance.

Constraint: $\mathbf{rtol}(i) \geq 0$ for all relevant i.

14: **atol**$(*)$ **– double array**

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.

The absolute local error tolerance.

Constraint: $\mathbf{atol}(i) \geq 0$ for all relevant i.

Note: corresponding elements of **rtol** and **atol** cannot both be 0.0.

15: **itol – int32 scalar**

A value to indicate the form of the local error test. **itol** indicates to d03pr 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:

itol	**rtol**	**atol**	w_i
1	scalar	scalar	$\mathbf{rtol}(1) \times \lvert\mathbf{u}(i)\rvert + \mathbf{atol}(1)$
2	scalar	vector	$\mathbf{rtol}(1) \times \lvert\mathbf{u}(i)\rvert + \mathbf{atol}(i)$

3	vector	scalar	$\mathbf{rtol}(i) \times \lvert\mathbf{u}(i)\rvert + \mathbf{atol}(1)$
4	vector	vector	$\mathbf{rtol}(i) \times \lvert\mathbf{u}(i)\rvert + \mathbf{atol}(i)$

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

The choice of norm used is defined by the parameter **norm_p**.

Constraint: $1 \leq \mathbf{itol} \leq 4$.

16: **norm_p – string**

The type of norm to be used.

norm_p = 'M'

Maximum norm.

norm_p = '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

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

while for the maximum norm

$$U_{\text{norm}} = \max_i \lvert\mathbf{u}(i)/w_i\rvert.$$

See the description of **itol** for the formulation of the weight vector w.

Constraint: **norm_p** = 'M' or 'A'.

17: **laopt – string**

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).

18: **algopt(30) – double array**

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 $j = 1, 2, \ldots,$ **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. $0.51 \leq$ **algopt**(5) ≤ 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 functions 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) $= 0.0001$.

19: **remesh – logical scalar**

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 d03pr. Remeshing can be switched off or on at specified times by using appropriate values for the parameters **nrmesh** and **trmesh** at each call.

20: **xfix**$(*)$ **– double array**

Note: the dimension of the array **xfix** must be at least max(1, **nxfix**).

xfix(i), for $i = 1, 2, \ldots,$ **nxfix**, must contain the value of the x co-ordinate at the ith fixed mesh point.

Constraint: **xfix**$(i) <$ **xfix**$(i+1)$, for $i = 1, 2, \ldots,$ **nxfix** $- 1$, and each fixed mesh point must coincide with a user-supplied initial mesh point, that is **xfix**$(i) =$ **x**(j) for some j, $2 \leq j \leq$ **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.

21: **nrmesh – int32 scalar**

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 d03pr to give greater flexibility over the times of remeshing.

22: **dxmesh – double scalar**

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

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

or

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

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

Constraint: **dxmesh** ≥ 0.0.

23: **trmesh – double scalar**

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 d03pr to force remeshing at several specified times.

24: **ipminf – int32 scalar**

The level of trace information regarding the adaptive remeshing. Details are directed to the current advisory message unit (see x04ab).

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.

25: **monitf – string containing name of m-file**

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 string `'d03pel'` may be used for **monitf**. **d03pel** is included in the NAG Fortran Library.

Its specification is:

```
[fmon] = monitf(t, npts, npde, x, u)
```

Input Parameters

1: **t – double scalar**

The current value of the independent variable t.

2: **npts – int32 scalar**

The number of mesh points in the interval $[a, b]$.

3: **npde – int32 scalar**

The number of PDEs in the system.

4: **x(npts) – double array**

The current mesh. **x**(i) contains the value of x_i, for $i = 1, 2, \ldots,$ **npts**.

5: **u(npde,npts) – double array**

The first dimension of the array **u** must be at least **neqn** = **npde** × **npts** + **ncode**

The second dimension of the array must be at least **npde** × **npts**

u(i,j) contains the value of $U_i(x,t)$ at $x =$ **x**(j) and time t, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**.

Output Parameters

1: **fmon(npts) – double array**

fmon(i) must contain the value of the monitor function $F^{\mathrm{mon}}(x)$ at mesh point $x =$ **x**(i).

26: **rsave(lrsave) – double array**

If **ind** $= 0$, **rsave** need not be set on entry.

If **ind** $= 1$, **rsave** must be unchanged from the previous call to the function because it contains required information about the iteration.

27: **isave(lisave) – int32 array**

If **ind** $= 0$, **isave** need not be set.

If **ind** $= 1$, **isave** must be unchanged from the previous call to the function 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.

28: **itask – int32 scalar**

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 $t =$ **tout**.

itask $= 4$

Normal computation of output values **u** at $t =$ **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: $1 \leq$ **itask** ≤ 5.

29: **itrace – int32 scalar**

The level of trace information required from d03pr 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 x04aa).

itrace $= 1$

Output from the underlying ODE solver is printed on the current advisory message unit (see x04ab). 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.

30: **ind – int32 scalar**

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 function. 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 d03pr.

Constraint: $0 \leq$ **ind** ≤ 1.

5.2 Optional Input Parameters

1: **npts – int32 scalar**

Default: The dimension of the array **x**.

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

Constraint: **npts** ≥ 3.

2: **nxi – int32 scalar**

Default: The dimension of the array **xi**.

The number of ODE/PDE coupling points.

Constraints:

if **ncode** $= 0$, **nxi** $= 0$;
if **ncode** > 0, **nxi** ≥ 0.

3: **neqn – int32 scalar**

Default: The dimension of the array **u**.

the number of ODEs in the time direction.

Constraint: **neqn** $=$ **npde** $\times$ **npts** $+$ **ncode**.

4: **nxfix – int32 scalar**

Default: The dimension of the array **xfix**.

The number of fixed mesh points.

Constraint: $0 \leq$ **nxfix** $\leq$ **npts** $- 2$

Note: the end points **x**(1) and **x**$($**npts**$)$ are fixed automatically and hence should not be specified as fixed points.

5: **xratio – double scalar**

Input bound on adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The remeshing functions will attempt to ensure that

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

Suggested value: **xratio** $= 1.5$.

Default: 1.5

Constraint: **xratio** > 1.0.

6: **con – double scalar**

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

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

(see Furzeland 1984). **con** gives you more control over the mesh distribution e.g., decreasing **con** allows more clustering. A typical value is $2/($**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** $= 2.0/($**npts** $- 1)$.

Default: $2.0/($**npts** $- 1)$

Constraint: $0.1/($**npts** $- 1) \leq$ **con** $\leq 10.0/($**npts** $- 1)$.

7: **lrsave – int32 scalar**

Default: The dimension of the array **rsave**.

Its size depends on the type of matrix algebra selected. If **laopt** = 'F', **lrsave** $\geq$ **neqn** $\times$ **neqn** + **neqn** + $NWKRES + LENODE$.

If **laopt** = 'B', **lrsave** $\geq (3 \times ML + MU + 2) \times$ **neqn** + $NWKRES + LENODE$.

If **laopt** = 'S', **lrsave** $\geq 4 \times$ **neqn** + $11 \times$ **neqn**$/2 + 1 + NWKRES + LENODE$.

Where

ML and MU are the lower and upper half bandwidths given by ML = **npde** + **nleft** $- 1$, and
$MU = 2 \times$ **npde** $-$ **nleft** $- 1$, for problems involving PDEs only, and
$ML = MU$ = **neqn** $- 1$, for coupled PDE/ODE problems.

$NWKRES$ = **npde** $\times (3 \times$ **npde** $+ 6 \times$ **nxi** + **npts** $+ 15)$ + **nxi** + **ncode** $+ 7 \times$ **npts** + **nxfix** $+ 1$, when **ncode** > 0 and **nxi** > 0, and
$NWKRES$ = **npde** $\times (3 \times$ **npde** + **npts** $+ 21)$ + **ncode** $+ 7 \times$ **npts** + **nxfix** $+ 2$,
when **ncode** > 0 and **nxi** $= 0$, and
$NWKRES$ = **npde** $\times (3 \times$ **npde** + **npts** $+ 21) + 7 \times$ **npts** + **nxfix** $+ 3$, when **ncode** $= 0$.

$LENODE = (6 + \text{int}($**algopt**$(2))) \times$ **neqn** $+ 50$, when the BDF method is used, and
$LENODE = 9 \times$ **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 function returns with **ifail** $= 15$.

8: **lisave – int32 scalar**

Default: The dimension of the array **isave**.

Its size depends on the type of matrix algebra selected:

if **laopt** = 'F', **lisave** ≥ 25 + **nxfix**;
if **laopt** = 'B', **lisave** $\geq$ **neqn** $+ 25$ + **nxfix**;
if **laopt** = 'S', **lisave** $\geq 25 \times$ **neqn** $+ 25$ + **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 function returns with **ifail** $= 15$.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **ts – double scalar**

The value of t corresponding to the solution values in **u**. Normally **ts** = **tout**.

2: **u(neqn) – double array**

u(**npde** $\times (j - 1) + i$) contains the computed solution $U_i(x_j, t)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**, and **u**(**npts** $\times$ **npde** $+ k$) contains $V_k(t)$, for $k = 1, 2, \ldots,$ **ncode**, evaluated at t = **ts**.

3: **x(npts) – double array**

The final values of the mesh points.

4: **rsave**(**lrsave**) **– double array**

If **ind** $= 0$, **rsave** need not be set on entry.

If **ind** $= 1$, **rsave** must be unchanged from the previous call to the function because it contains required information about the iteration.

5: **isave**(**lisave**) **– int32 array**

If **ind** $= 0$, **isave** need not be set.

If **ind** $= 1$, **isave** must be unchanged from the previous call to the function 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.

6: **ind – int32 scalar**

ind $= 1$.

7: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

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 [**x**(1), **x**(**npts**)],
or **npts** < 3,
or **npde** < 1,
or **nleft** not in the range 0 to **npde**,
or **norm_p** $\neq$ '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 d03pr,
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 function 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 function 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 $= 9$ (d02nn)

A serious error has occurred in an internal call to the specified function. 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 $(\textbf{atol}(i) = 0.0)$ 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 d03pr.

7 Accuracy

d03pr 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 (d03pp 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 (d03ps 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

d03pr_bndary.m

```
function [res, ires] = bndary(npde, t, ibnd, nobc, u, udot, ncode, v,
vdot, ires)
  res = zeros(nobc, 1);

  pp = 2*pi;
  if (ibnd == 0)
    if (ires ~= -1)
      res(1) = u(1) - ...
                        0.5*(exp(t)+exp(-3*t))  -  0.25*(sin(pp*9*t^2)-
sin(pp*t^2)) - 2*t^2;
    end
  else
    if (ires ~= -1)
      res(1) = u(2) - ...
                                (exp(1-3*t)-exp(1+t)   +0.5*(sin(pp*(1-
3*t)^2)+sin(pp*(1+t) ^2))+1+5*t^2-2*t);
    end
  end
```

d03pr_monitf.m

```
function [fmon] = monitf(t, npts, npde, x, u)
  fmon = zeros(npts, 1);

  for 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);
  end
  fmon(1) = fmon(2);
  fmon(npts) = fmon(npts-1);
```

d03pr_odedef.m

```
function  [f,  ires]  =  odedef(npde,  t,  ncode,  v,  vdot,  nxi,  xi,  ucp,
ucpx, ucpt, ires)
  f = zeros(ncode, 1);
```

d03pr_pdedef.m

```
function [res, ires] = pdedef(npde, t, x, u, udot, ux, ncode, v, vdot,
ires)
  res = zeros(npde, 1);

  if (ires == -1)
    res(1) = udot(1);
    res(2) = udot(2);
  else
    res(1) = udot(1) + ux(1) + ux(2);
    res(2) = udot(2) + 4.0d0*ux(1) + ux(2);
  end
```

d03pr_uvinit.m

```
function [u, v] = uvinit(npde, npts, nxi, x, xi, ncode)
  u = zeros(npde, npts);
  v = zeros(ncode, 1);

  for i = 1:npts
    u(1,i) = exp(x(i));
    u(2,i) = x(i)^2 + sin(2*pi*x(i)^2);
  end
```

```
npde = int32(2);
ts = 0;
tout = 0.05;
u = zeros(122, 1);
x = [0;
     0.01666666666666667;
     0.03333333333333333;
     0.05;
     0.06666666666666667;
     0.08333333333333333;
     0.1;
     0.1166666666666667;
     0.1333333333333333;
     0.15;
     0.1666666666666667;
     0.1833333333333333;
     0.2;
     0.2166666666666667;
     0.2333333333333333;
     0.25;
     0.2666666666666667;
     0.2833333333333333;
     0.3;
     0.3166666666666667;
     0.3333333333333333;
     0.35;
     0.3666666666666666;
     0.3833333333333334;
     0.4;
     0.4166666666666667;
     0.4333333333333333;
     0.45;
     0.4666666666666667;
     0.4833333333333333;
     0.5;
     0.5166666666666667;
     0.5333333333333333;
     0.55;
     0.5666666666666667;
     0.5833333333333334;
     0.6;
     0.6166666666666667;
     0.6333333333333333;
     0.65;
     0.6666666666666666;
     0.6833333333333333;
     0.7;
     0.7166666666666667;
     0.7333333333333333;
     0.75;
     0.7666666666666667;
     0.7833333333333333;
     0.8;
     0.8166666666666667;
     0.8333333333333334;
     0.85;
     0.8666666666666667;
     0.8833333333333333;
     0.9;
     0.9166666666666666;
     0.9333333333333333;
     0.95;
     0.9666666666666667;
     0.9833333333333333;
     1];
nleft = int32(1);
ncode = int32(0);
xi = [];
rtol = [5e-05];
atol = [5e-05];
itol = int32(1);
normtype = 'A';
laopt = 'F';
algopt = zeros(30, 1);
remesh = true;
xfix = [];
nrmesh = int32(3);
dxmesh = 0;
trmesh = 0;
ipminf = int32(0);
rsave = zeros(17004, 1);
isave = zeros(25, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
[tsOut, uOut, xOut, rsaveOut, isaveOut, indOut, ifail] = ...
    d03pr(npde, ts, tout, 'd03pr_pdedef', 'd03pr_bndary', 'd03pr_uvinit',
...
     u, x, nleft, ncode, 'd03pr_odedef', xi, rtol, atol, itol, normtype,
...
    laopt, algopt, remesh, xfix, nrmesh, dxmesh, trmesh, ipminf, ...
      'd03pr_monitf', rsave, isave, itask, itrace, ind, 'xratio', 1.2,
'con', 5/60)
```

```
tsOut =
    0.0500
uOut =
     array elided
xOut =
     array elided
rsaveOut =
     array elided
isaveOut =
          18
        1155
           9
           3
          46
          46
           0
         114
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
indOut =
                      1
ifail =
                      0
```