NAG Toolbox for MATLAB

d03pf

1 Purpose

d03pf integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms. The system must be posed in conservative form. Convection terms are discretized using a sophisticated upwind scheme involving a user-supplied numerical flux function based on the solution of a Riemann problem at each mesh point. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs), and the resulting system is solved using a backward differentiation formula (BDF) method.

2 Syntax

```
[ts, u, rsave, isave, ind, ifail] = d03pf(ts, tout, pdedef, numflx,
bndary, u, x, acc, tsmax, rsave, isave, itask, itrace, ind, 'npde',
npde, 'npts', npts, 'lrsave', lrsave, 'lisave', lisave)
```

3 Description

d03pf integrates the system of convection-diffusion equations in conservative form:

$$\sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + \frac{\partial F_i}{\partial x} = C_i\frac{\partial D_i}{\partial x} + S_i, \tag{1}$$

or the hyperbolic convection-only system:

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial x} = 0, \tag{2}$$

for $i = 1, 2, \ldots, \mathbf{npde}$, $a \le x \le b$, $t \ge t_0$, where the vector U is the set of solution values

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

The functions $P_{i,j}$, F_i, C_i and S_i depend on x, t and U; and D_i depends on x, t, U and U_x, where U_x is the spatial derivative of U. Note that $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives; and none of the functions may depend on time derivatives. In terms of conservation laws, F_i, $\dfrac{C_i\partial D_i}{\partial x}$ and S_i are the convective flux, diffusion and source terms respectively.

The integration in time is from t_0 to t_{out}, over the space interval $a \le x \le b$, where $a = x_1$ and $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$. The initial values of the functions $U(x,t)$ must be given at $t = t_0$.

The PDEs are approximated by a system of ODEs in time for the values of U_i at mesh points using a spatial discretization method similar to the central-difference scheme used in d03pc, d03ph and d03pp, but with the flux F_i replaced by a *numerical flux*, which is a representation of the flux taking into account the direction of the flow of information at that point (i.e., the direction of the characteristics). Simple central differencing of the numerical flux then becomes a sophisticated upwind scheme in which the correct direction of upwinding is automatically achieved.

The numerical flux vector, $\hat{F}_i$ say, must be calculated by you in terms of the *left* and *right* values of the solution vector U (denoted by U_L and U_R respectively), at each mid-point of the mesh $x_{j-1/2} = \left(x_{j-1} + x_j\right)/2$, for $j = 2, 3, \ldots, \mathbf{npts}$. The left and right values are calculated by d03pf from two adjacent mesh points using a standard upwind technique combined with a Van Leer slope-limiter (see LeVeque 1990). The physically correct value for $\hat{F}_i$ is derived from the solution of the Riemann problem given by

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial y} = 0, \tag{3}$$

where $y = x - x_{j-1/2}$, i.e., $y = 0$ corresponds to $x = x_{j-1/2}$, with discontinuous initial values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$, using an *approximate Riemann solver*. This applies for either of the systems (1) or (2); the numerical flux is independent of the functions $P_{i,j}$, C_i, D_i and S_i. A description of several approximate Riemann solvers can be found in LeVeque 1990 and Berzins *et al.* 1989. Roe's scheme (see Roe 1981) is perhaps the easiest to understand and use, and a brief summary follows. Consider the system of PDEs $U_t + F_x = 0$ or equivalently $U_t + AU_x = 0$. Provided the system is linear in U, i.e., the Jacobian matrix A does not depend on U, the numerical flux $\hat{F}$ is given by

$$\hat{F} = \tfrac{1}{2}(F_L + F_R) - \tfrac{1}{2}\sum_{k=1}^{\mathbf{npde}} \alpha_k |\lambda_k| e_k, \tag{4}$$

where F_L (F_R) is the flux F calculated at the left (right) value of U, denoted by U_L (U_R); the λ_k are the eigenvalues of A; the e_k are the right eigenvectors of A; and the α_k are defined by

$$U_R - U_L = \sum_{k=1}^{\mathbf{npde}} \alpha_k e_k. \tag{5}$$

An example is given in Section 9.

If the system is nonlinear, Roe's scheme requires that a linearized Jacobian is found (see Roe 1981).

The functions $P_{i,j}$, C_i, D_i and S_i (but **not** F_i) must be specified in a (sub)program **pdedef**. The numerical flux $\hat{F}_i$ must be supplied in a separate user-supplied (sub)program **numflx**. For problems in the form (2), the actual argument **d03pfp** may be used for **pdedef**. **d03pfp** is included in the NAG Fortran Library and sets the matrix with entries $P_{i,j}$ to the identity matrix, and the functions C_i, D_i and S_i to zero.

The boundary condition specification has sufficient flexibility to allow for different types of problems. For second-order problems, i.e., D_i depending on U_x, a boundary condition is required for each PDE at both boundaries for the problem to be well-posed. If there are no second-order terms present, then the continuous PDE problem generally requires exactly one boundary condition for each PDE, that is **npde** boundary conditions in total. However, in common with most discretization schemes for first-order problems, a *numerical boundary condition* is required at the other boundary for each PDE. In order to be consistent with the characteristic directions of the PDE system, the numerical boundary conditions must be derived from the solution inside the domain in some manner (see below). You must supply both types of boundary conditions, i.e., a total of **npde** conditions at each boundary point.

The position of each boundary condition should be chosen with care. In simple terms, if information is flowing into the domain then a physical boundary condition is required at that boundary, and a numerical boundary condition is required at the other boundary. In many cases the boundary conditions are simple, e.g., for the linear advection equation. In general you should calculate the characteristics of the PDE system and specify a physical boundary condition for each of the characteristic variables associated with incoming characteristics, and a numerical boundary condition for each outgoing characteristic.

A common way of providing numerical boundary conditions is to extrapolate the characteristic variables from the inside of the domain. Note that only linear extrapolation is allowed in this function (for greater flexibility the function d03pl should be used). For problems in which the solution is known to be uniform (in space) towards a boundary during the period of integration then extrapolation is unnecessary; the numerical boundary condition can be supplied as the known solution at the boundary. Examples can be found in Section 9.

The boundary conditions must be specified in a user-supplied (sub)program **bndary** in the form

$$G_i^L(x,t,U) = 0 \qquad \text{at } x = a, \qquad i = 1,2,\ldots,\mathbf{npde}, \tag{6}$$

at the left-hand boundary, and

$$G_i^R(x,t,U) = 0 \qquad \text{at } x = b, \qquad i = 1,2,\ldots,\mathbf{npde}, \tag{7}$$

at the right-hand boundary.

Note that spatial derivatives at the boundary are not passed explicitly to the user-supplied (sub)program **bndary**, but they can be calculated using values of U at and adjacent to the boundaries if required. However, it should be noted that instabilities may occur if such one-sided differencing opposes the characteristic direction at the boundary.

The problem is subject to the following restrictions:

(i) $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives;

(ii) $P_{i,j}$, F_i, C_i, D_i and S_i must not depend on any time derivatives;

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

(iv) The evaluation of the terms $P_{i,j}$, C_i, D_i and S_i is done by calling the (sub)program **pdedef** at a point approximately midway between each pair of mesh points in turn. Any discontinuities in these functions **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathbf{npts}}$;

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

In total there are **npde** $\times$ **npts** ODEs in the time direction. This system is then integrated forwards in time using a BDF method.

For further details of the algorithm, see Pennington and Berzins 1994 and the references therein.

4 References

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

Hirsch C 1990 *Numerical Computation of Internal and External Flows, Volume 2: Computational Methods for Inviscid and Viscous Flows* John Wiley

LeVeque R J 1990 *Numerical Methods for Conservation Laws* Birkhäuser Verlag

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

Roe P L 1981 Approximate Riemann solvers, parameter vectors, and difference schemes *J. Comput. Phys.* **43** 357–372

5 Parameters

5.1 Compulsory Input Parameters

1: **ts – double scalar**

The initial value of the independent variable t.

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

2: **tout – double scalar**

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

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

pdedef must evaluate the functions $P_{i,j}$, C_i, D_i and S_i which partially define the system of PDEs. $P_{i,j}$, C_i and S_i may depend on x, t and U; D_i may depend on x, t, U and U_x. **pdedef** is called approximately midway between each pair of mesh points in turn by d03pf. The actual argument **d03pfp** may be used for **pdedef** for problems in the form (2). **d03pfp** is included in the NAG Fortran Library.

Its specification is:

```
[p, c, d, s, ires] = pdedef(npde, t, x, u, ux, 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 $i = 1,2,\ldots,$ **npde**.

5: **ux(npde) – double array**

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

6: **ires – int32 scalar**

Set to -1 or 1.

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 d03pf returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **p(npde,npde) – double array**

p(i,j) must be set to the value of $P_{i,j}(x,t,U)$, for $i,j = 1,2,\ldots,$ **npde**.

2: **c(npde) – double array**

c(i) must be set to the value of $C_i(x,t,U)$, for $i = 1,2,\ldots,$ **npde**.

3: **d(npde) – double array**

d(i) must be set to the value of $D_i(x,t,U,U_x)$, for $i = 1,2,\ldots,$ **npde**.

4: **s(npde) – double array**

s(i) must be set to the value of $S_i(x,t,U)$, for $i = 1,2,\ldots,$ **npde**.

5: **ires – int32 scalar**

Set to -1 or 1.

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 d03pf returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

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

numflx must supply the numerical flux for each PDE given the *left* and *right* values of the solution vector **u**. **numflx** is called approximately midway between each pair of mesh points in turn by d03pf.

Its specification is:

```
[flux, ires] = numflx(npde, t, x, uleft, uright, 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: **uleft**(**npde**) **– double array**

uleft(i) contains the *left* value of the component $U_i(x)$, for $i = 1, 2, \ldots,$ **npde**.

5: **uright**(**npde**) **– double array**

uright(i) contains the *right* value of the component $U_i(x)$, for $i = 1, 2, \ldots,$ **npde**.

6: **ires – int32 scalar**

Set to -1 or 1.

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 d03pf returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

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

flux(i) must be set to the numerical flux $\hat{F}_i$, for $i = 1, 2, \ldots,$ **npde**.

2: **ires** – **int32 scalar**

Set to -1 or 1.

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 d03pf 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 physical and numerical boundary conditions, as given by (6) and (7).

Its specification is:

```
[g, ires] = bndary(npde, npts, t, x, u, ibnd, ires)
```

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: **t** – **double scalar**

The current value of the independent variable t.

4: **x**(**npts**) – **double array**

The mesh points in the spatial direction. **x**(1) corresponds to the left-hand boundary, a, and **x**(**npts**) corresponds to the right-hand boundary, b.

5: **u(npde,3) – double array**

Contains the value of solution components in the boundary region.

If **ibnd** $= 0$, $\mathbf{u}(i,j)$ contains the value of the component $U_i(x,t)$ at $x = \mathbf{x}(j)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,3$.

If **ibnd** $\neq 0$, $\mathbf{u}(i,j)$ contains the value of the component $U_i(x,t)$ at $x = \mathbf{x}(\mathbf{npts} - j + 1)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,3$.

6: **ibnd – int32 scalar**

Specifies which boundary conditions are to be evaluated.

ibnd $= 0$

bndary must evaluate the left-hand boundary condition at $x = a$.

ibnd $\neq 0$

bndary must evaluate the right-hand boundary condition at $x = b$.

7: **ires – int32 scalar**

Set to -1 or 1.

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 d03pf returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **g(npde) – double array**

$\mathbf{g}(i)$ must contain the ith component of either $\mathbf{g}^L$ or $\mathbf{g}^R$ in (6) and (7), depending on the value of **ibnd**, for $i = 1,2,\ldots,$ **npde**.

2: **ires – int32 scalar**

Set to -1 or 1.

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 d03pf returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

6: **u(npde,npts) – double array**

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

7: **x(npts) – double array**

The 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**).

8: **acc(2) – double array**

The components of **acc** contain the relative and absolute error tolerances used in the local error test in the time integration.

If $\mathrm{E}(i,j)$ is the estimated error for U_i at the jth mesh point, the error test is

$$\mathrm{E}(i,j) = \textbf{acc}(1) \times \textbf{u}(i,j) + \textbf{acc}(2).$$

Constraint: **acc**(1) and **acc**$(2) \geq 0.0$ (but not both zero).

9: **tsmax – double scalar**

The maximum absolute step size to be allowed in the time integration. If **tsmax** $= 0.0$ then no maximum is imposed.

Constraint: **tsmax** ≥ 0.0.

10: **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.

11: **isave(lisave) – int32 array**

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

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

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 computing 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 last backward differentiation formula method used.

isave(5)

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

12: **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**.

Constraint: $1 \leq$ **itask** ≤ 3.

13: **itrace – int32 scalar**

The level of trace information required from d03pf and the underlying ODE solver. **itrace** may take the value -1, 0, 1, 2 or 3.

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 > 0

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.

If **itrace** < -1, then -1 is assumed and similarly if **itrace** > 3, then 3 is assumed.

The advisory messages are given in greater detail as **itrace** increases. You are advised to set **itrace** $= 0$, unless you are experienced with sub-chapter D02M/N.

14: **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** should be reset between calls to d03pf.

Constraint: $0 \leq$ **ind** ≤ 1.

5.2 Optional Input Parameters

1: **npde – int32 scalar**

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

the number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **npts – int32 scalar**

Default: The dimension of the arrays **u**, **x**. (An error is raised if these dimensions are not equal.)

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

Constraint: **npts** ≥ 3.

3: **lrsave – int32 scalar**

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

Constraint: **lrsave** $\geq (11 + 9 \times$ **npde**$) \times$ **npde** $\times$ **npts** $+ (32 + 3 \times$ **npde**$) \times$ **npde** $+ 7 \times$ **npts** $+ 54$.

4: **lisave – int32 scalar**

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

Constraint: **lisave** $\geq$ **npde** $\times$ **npts** $+ 24$.

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**(**npde**,**npts**) **– double array**

u(i,j) will contain the computed solution $U_i(x,t)$ at $x =$ **x**(j) and $t =$ **ts**, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**.

3: **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.

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

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

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

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 computing 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 last backward differentiation formula method used.

isave(5)

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

5: **ind – int32 scalar**

ind $= 1$.

6: **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, **ts** $\geq$ **tout**,
or **tout** $-$ **ts** is too small,
or **itask** $= 1$, 2 or 3,
or **npts** < 3,
or **npde** < 1,
or **ind** $\neq 0$ or 1,
or incorrectly defined user mesh, i.e., $\mathbf{x}(i) \geq \mathbf{x}(i+1)$ for some $i = 1, \ldots,$ **npts** $- 1$,
or **lrsave** or **lisave** are too small,
or **ind** $= 1$ on initial entry to d03pf,
or **acc**(1) or **acc**$(2) < 0.0$,
or **acc**(1) or **acc**(2) are both zero,
or **tsmax** < 0.0.

ifail $= 2$

The underlying ODE solver cannot make any further progress, with the values of **acc**, 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 specification 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**, **numflx** or **bndary** when the residual in the underlying ODE solver was being evaluated. Incorrect specification of boundary conditions may also result in this error.

ifail $= 5$

In solving the ODE system, a singular Jacobian has been encountered. Check the problem formulation.

ifail $= 6$

When evaluating the residual in solving the ODE system, **ires** was set to 2 in at least one of the user-supplied (sub)programs **pdedef**, **numflx** or **bndary**. Integration was successful as far as $t =$ **ts**.

ifail $= 7$

The values of **acc**(1) and **acc**(2) 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**, **numflx** or **bndary**, **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 and 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 the values of **acc** 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** $\neq 2$.)

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 when **itrace** ≥ 1).

ifail $= 12$

Not applicable.

ifail $= 13$

Not applicable.

ifail $= 14$

One or more of the functions $P_{i,j}$, D_i or C_i was detected as depending on time derivatives, which is not permissible.

7 Accuracy

d03pf 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 components of the accuracy parameter, **acc**.

8 Further Comments

d03pf is designed to solve systems of PDEs in conservative form, with optional source terms which are independent of space derivatives, and optional second-order diffusion terms. The use of the function to solve systems which are not naturally in this form is discouraged, and you are advised to use one of the central-difference schemes for such problems.

You should be aware of the stability limitations for hyperbolic PDEs. For most problems with small error tolerances the ODE integrator does not attempt unstable time steps, but in some cases a maximum time step should be imposed using **tsmax**. It is worth experimenting with this parameter, particularly if the integration appears to progress unrealistically fast (with large time steps). Setting the maximum time step to the minimum mesh size is a safe measure, although in some cases this may be too restrictive.

Problems with source terms should be treated with caution, as it is known that for large source terms stable and reasonable looking solutions can be obtained which are in fact incorrect, exhibiting non-physical speeds of propagation of discontinuities (typically one spatial mesh point per time step). It is essential to employ a very fine mesh for problems with source terms and discontinuities, and to check for non-physical propagation speeds by comparing results for different mesh sizes. Further details and an example can be found in Pennington and Berzins 1994.

The time taken depends on the complexity of the system and on the accuracy requested.

9 Example

d03pf_bndary.m

```
function [g, ires] = bndary(npde, npts, t, x, u, ibnd, ires)
      g = zeros(npde,1);
      if (ibnd == 0)
         ue = exact(t,npde,x(1),1);
         c = (x(2)-x(1))/(x(3)-x(2));
         exu1 = (1+c)*u(1,2) - c*u(1,3);
         exu2 = (1+c)*u(2,2) - c*u(2,3);
         g(1) = 2*u(1,1) + u(2,1) - 2*ue(1,1) - ue(2,1);
         g(2) = 2*u(1,1) - u(2,1) - 2*exu1 + exu2;
      else
         ue = exact(t,npde,x(npts),1);
         c = (x(npts)-x(npts-1))/(x(npts-1)-x(npts-2));
         exu1 = (1+c)*u(1,2) - c*u(1,3);
         exu2 = (1+c)*u(2,2) - c*u(2,3);
         g(1) = 2*u(1,1) - u(2,1) - 2*ue(1,1) + ue(2,1);
         g(2) = 2*u(1,1) + u(2,1) - 2*exu1 - exu2;
      end

% exact solution (for comparison and b.c. purposes)
function u = exact(t,npde,x,npts)
  u = zeros(npde,npts);
  for i = 1:npts
    x1 = x(i) + t;
    x2 = x(i) - 3*t;
                        u(1,i)=0.5*(exp(x1)+exp(x2))+0.25*(sin(2*pi*x2^2)-
sin(2*pi*x1^2))+2*t^2-2*x(i)*t;
                                                    u(2,i)=exp(x2)-
exp(x1)+0.5*(sin(2*pi*x2^2)+sin(2*pi*x1^2))+x(i)^2+5*t^2-2*x(i)*t;
  end
```

d03pf_numflx.m

```
function [flux,ires] = numflx(npde, t, x, uleft, uright, ires)
      flux = zeros(npde,1);
```

```
      flux(1) = 0.5*(-uright(1)+3*uleft(1)+0.5*uright(2)+1.5*uleft(2));
      flux(2) = 0.5*(2*uright(1)+6*uleft(1)-uright(2)+3*uleft(2));
```

d03pf_pdedef.m

```
function [p, c, d, s, ires] = pdedef(npde, t, x, u, ux, ires)
    %  PDEDEF routine for simple hyperbolic system, equivalent to
    %  routine D03PFP
    p = zeros(npde, npde);
    c = zeros(npde, 1);
    d = zeros(npde, 1);
    s = zeros(npde, 1);
    for i = 1:npde
      p(i,i) = 1;
    end
```

```
ts = 0;
tout = 0.1;
u = [1, 1.010050167084168, 1.020201340026756, 1.030454533953517, ...
    1.040810774192388, 1.051271096376024, 1.06183654654536, ...
    1.072508181254217, 1.083287067674959, 1.09417428370521, ...
    1.105170918075648, 1.116278070458871, 1.127496851579376, ...
    1.138828383324622, 1.150273798857227, 1.161834242728283, ...
    1.17351087099181, 1.185304851320365, 1.19721736312181, ...
    1.209249597657251, 1.22140275816017, 1.233678059956743, ...
    1.246076730587381, 1.258600009929478, 1.271249150321405, ...
    1.284025416687741, 1.296930086665772, 1.309964450733247, ...
    1.323129812337437, 1.336427488025472, 1.349858807576003, ...
    1.363425114132178, 1.377127764335957, 1.39096812846378, ...
    1.404947590563594, 1.419067548593257, 1.43332941456034, ...
    1.447734614663325, 1.462284589434224, 1.476980793882643, ...
    1.49182469764127, 1.506817785112853, 1.521961555618634, ...
    1.537257523548281, 1.552707218511336, 1.568312185490169, ...
    1.584073984994482, 1.59999419321736, 1.616074402192893, ...
    1.632316219955379, 1.648721270700128, 1.665291194945886, ...
    1.682027649698886, 1.698932308618551, 1.716006862184859, ...
    1.733253017867395, 1.750672500296101, 1.768267051433735, ...
    1.786038430750073, 1.803988415397857, 1.822118800390509, ...
    1.840431398781637, 1.858928041846342, 1.877610579264343, ...
    1.896480879304951, 1.915540829013896, 1.934792334402032, ...
    1.95423732063594, 1.973877732230448, 1.993715533243082, ...
    2.013752707470477, 2.033991258646751, 2.054433210643888, ...
    2.075080607674122, 2.095935514494364, 2.117000016612675, ...
    2.138276220496818, 2.159766253784915, 2.181472265498201, ...
    2.203396426255937, 2.225540928492468, 2.247907986676472, ...
    2.270499837532406, 2.293318740264183, 2.316366976781091, ...
    2.339646851925991, 2.363160693705795, 2.386910853524276, ...
    2.41089970641721, 2.435129651289874, 2.45960311115695, ...
    2.484322533384816, 2.509290389936298, 2.534509177617855, ...
    2.559981418329271, 2.585709659315846, 2.611696473423118, ...
    2.637944459354153, 2.664456241929417, 2.691234472349262, ...
    2.718281828459045;
                      0,    0.0007283184893761486,    0.002913271477003692,
0.006554836638408964, ...
    0.01165292715673064, 0.01820731731182075, 0.02621753831672977, ...
    0.03568274442576258, 0.04660154936295328, 0.05897183315779799, ...
    0.07279051952931347, 0.08805332403289762, 0.1047544732799356, ...
    0.1228863956604543, 0.1424393841471642, 0.1634012319375826, ...
    0.1857568419021818, 0.2094878110529674, 0.2345719915307735, ...
    0.2609830299326747, 0.2886898871648548, 0.3176563414121236, ...
    0.3478404772637143, 0.3791941645260716, 0.411662530786469, ...
    0.4451834323650897, 0.4796869289054211, 0.515094767500187, ...
    0.5513198829282125, 0.588265921280956, 0.6258267949789966, ...
    0.6638862779100518, 0.702317650151088, 0.7409834024559822, ...
    0.7797350113834701, 0.8184127965923141, 0.8568458724243957, ...
    0.8948522064124793, 0.9322387977664519, 0.9688019891867994, ...
```

```
    1.004327925502015, 1.038593172601059, 1.071365509904915, ...
    1.102404909163783, 1.131464711648689, 1.15829301479833, ...
    1.182634278054065, 1.204231155939365, 1.222826564387686, ...
    1.238165983870076, 1.25, 1.258087078981321, ...
    1.26219657150561, 1.26211193449322, 1.257634155413497, ...
    1.248585358827545, 1.234812569297339, 1.216191598935723, ...
    1.192631021684547, 1.164076189969761, 1.130513242775789, ...
    1.09197304750735, 1.04853501138347, 1.000330691672591, ...
    0.9475471279847715, 0.8904298142605735, 0.8292852232270232, ...
    0.764482792135553, 0.6964562757751382, 0.6257043712966427, ...
    0.5527905195293141, 0.4783417894574812, 0.403046756591692, ...
    0.3276522923413058, 0.2529591903857135, 0.1798165676349105, ...
    0.1091149918194066, 0.0417783051635255, -0.02124586596926006, ...
    -0.07899690212315057, -0.1305132427757891, -0.1748456304888814, ...
    -0.2110714466863253, -0.2383100047810946, -0.2557386309271434, ...
    -0.2626093261547741, -0.2582657667668944, -0.2421603634586167, ...
    -0.2138710646187889, -0.1731175567333843, -0.1197764858882511, ...
    -0.05389530029404987, 0.02429570418093795, 0.1143735639632994, ...
    0.2157154834078929, 0.327494747956721, 0.4486801170717872, ...
    0.5780390672923718, 0.7141452106973597, 0.855390155185901, ...
    0.9999999999999996];
x = [0;
     0.01;
     0.02;
     0.03;
     0.04;
     0.05;
     0.06;
     0.07000000000000001;
     0.08;
     0.09;
     0.1;
     0.11;
     0.12;
     0.13;
     0.14;
     0.15;
     0.16;
     0.17;
     0.18;
     0.19;
     0.2;
     0.21;
     0.22;
     0.23;
     0.24;
     0.25;
     0.26;
     0.27;
     0.28;
     0.29;
     0.3;
     0.31;
     0.32;
     0.33;
     0.34;
     0.35;
     0.36;
     0.37;
     0.38;
     0.39;
     0.4;
     0.41;
     0.42;
     0.43;
     0.44;
     0.45;
     0.46;
     0.47;
     0.48;
```

```
     0.49;
     0.5;
     0.51;
     0.52;
     0.53;
     0.54;
     0.55;
     0.5600000000000001;
     0.57;
     0.58;
     0.59;
     0.6;
     0.61;
     0.62;
     0.63;
     0.64;
     0.65;
     0.66;
     0.67;
     0.68;
     0.6899999999999999;
     0.7;
     0.71;
     0.72;
     0.73;
     0.74;
     0.75;
     0.76;
     0.77;
     0.78;
     0.79;
     0.8;
     0.8100000000000001;
     0.82;
     0.83;
     0.84;
     0.85;
     0.86;
     0.87;
     0.88;
     0.89;
     0.9;
     0.91;
     0.92;
     0.93;
     0.9399999999999999;
     0.95;
     0.96;
     0.97;
     0.98;
     0.99;
     1];
acc = [0.0001;
     1e-05];
tsmax = 0;
rsave = zeros(6695, 1);
isave = zeros(226, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
[tsOut, uOut, rsaveOut, isaveOut, indOut, ifail] = ...
    d03pf(ts, tout, 'd03pf_pdedef', 'd03pf_numflx', 'd03pf_bndary', u, x,
acc, ...
    tsmax, rsave, isave, itask, itrace, ind)
```

```
tsOut =
    0.1000
uOut =
     array elided
rsaveOut =
```

```
     array elided
isaveOut =
     array elided
indOut =
           1
ifail =
           0
```