NAG Toolbox for MATLAB

d03pl

1 Purpose

d03pl integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms and scope for coupled ordinary differential equations (ODEs). 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 partial differential equations (PDEs) to a system of ODEs, and the resulting system is solved using a backward differentiation formula (BDF) method or a Theta method.

2 Syntax

```
[ts, u, rsave, isave, ind, ifail] = d03pl(npde, ts, tout, pdedef,
numflx, bndary, u, x, ncode, odedef, xi, rtol, atol, itol, norm_p,
laopt, algopt, rsave, isave, itask, itrace, ind, 'npts', npts, 'nxi',
nxi, 'neqn', neqn, 'lrsave', lrsave, 'lisave', lisave)
```

3 Description

d03pl 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 $i=1,2,\dots,\mathbf{npde}$, $\quad a\le x\le b,\quad t\ge t_0$, where the vector U is the set of PDE solution values

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

The optional coupled ODEs are of the general form

$$R_i\left(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*\right)=0,\qquad i=1,2,\dots,\mathbf{ncode}, \tag{3}$$

where the vector V is the set of ODE solution values

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

$\dot{V}$ denotes its derivative with respect to time, and U_x is the spatial derivative of U.

In (1), $P_{i,j}$, F_i and C_i depend on x, t, U and V; D_i depends on x, t, U, U_x and V; and S_i depends on x, t, U, V and **linearly** on $\dot{V}$. Note that $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives, and $P_{i,j}$, F_i, C_i and D_i must not depend on any 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.

In (3), 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 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 R_i may depend only linearly on time derivatives. Hence (3) may be written more precisely as

$$R=L-M\dot{V}-NU_t^*, \tag{4}$$

where $R = \left[R_1, \ldots, R_{\mathbf{ncode}}\right]^{\mathrm{T}}$, L is a vector of length **ncode**, M is an **ncode** by **ncode** matrix, N is an **ncode** by $\left(n_\xi \times \mathbf{npde}\right)$ matrix and the entries in L, M and N 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 L, M and N. (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 user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$. The initial values of the functions $U(x,t)$ and $V(t)$ 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 $x_{j-\frac{1}{2}} = \left(x_{j-1} + x_j\right)/2$, for $j = 2, 3, \ldots, \mathbf{npts}$. The left and right values are calculated by d03pl 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

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial y} = 0, \tag{5}$$

where $y = x - x_{j-\frac{1}{2}}$, i.e., $y = 0$ corresponds to $x = x_{j-\frac{1}{2}}$, 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

$$\hat{F} = \tfrac{1}{2}(F_L + F_R) - \tfrac{1}{2}\sum_{k=1}^{\mathbf{npde}} \alpha_k |\lambda_k| e_k, \tag{6}$$

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

$$U_R - U_L = \sum_{k=1}^{\mathbf{npde}} \alpha_k e_k. \tag{7}$$

An example is given in Section 9 and in the d03pf documentation.

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 **d03plp** may be used for **pdedef**. **d03plp** 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 condition, 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 when using banded matrix algebra the fixed bandwidth means that only linear extrapolation is allowed, i.e., using information at just two interior points adjacent to the boundary). 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. Another method of supplying numerical boundary conditions involves the solution of the characteristic equations associated with the outgoing characteristics. Examples of both methods can be found in Section 9 and in the d03pf documentation.

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

$$G_i^L\left(x,t,U,V,\dot{V}\right)=0 \qquad \text{at } x=a, \qquad i=1,2,\ldots,\mathbf{npde}, \tag{8}$$

at the left-hand boundary, and

$$G_i^R\left(x,t,U,V,\dot{V}\right)=0 \qquad \text{at } x=b, \qquad i=1,2,\ldots,\mathbf{npde}, \tag{9}$$

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 algebraic-differential equation system which is defined by the functions R_i must be specified in a (sub)program **odedef**. You must also specify the coupling points ξ (if any) in the array **xi**.

The problem is subject to the following restrictions:

(i) In (1), $\dot{V}_j(t)$, for $j=1,2,\ldots,\mathbf{ncode}$, may only appear **linearly** in the functions S_i, for $i=1,2,\ldots,\mathbf{npde}$, with a similar restriction for G_i^L and G_i^R;

(ii) $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives; and $P_{i,j}$, F_i, C_i and D_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 $\mathbf{npde} \times \mathbf{npts} + \mathbf{ncode}$ ODEs in the time direction. This system is then integrated forwards in time using a BDF or Theta method, optionally switching between Newton's method and functional iteration (see Berzins *et al.* 1989).

For further details of the scheme, 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

Sod G A 1978 A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws *J. Comput. Phys.* **27** 1–31

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 $P_{i,j}$, C_i, D_i and S_i which partially define the system of PDEs. $P_{i,j}$ and C_i may depend on x, t, U and V; D_i may depend on x, t, U, U_x and V; and S_i may depend on x, t, U, V and linearly on $\dot{V}$. **pdedef** is called approximately midway between each pair of mesh points in turn by d03pl. The actual argument **d03plp** may be used for **pdedef** for problems in the form (2). **d03plp** is included in the NAG Fortran Library.

Its specification is:

```
[p, c, d, s, ires] = pdedef(npde, t, x, u, 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 $i = 1, 2, \ldots,$ **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: **ncode – int32 scalar**

The number of coupled ODEs in the system.

7: **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**.

8: **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: $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**, may only appear linearly in S_j, for $j = 1, 2, \ldots,$ **npde**.

9: **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 d03pl 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 $P_{i,j}(x,t,U,V)$, for $i,j = 1, 2, \ldots,$ **npde**.

2: **c(npde) – double array**

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

3: **d(npde) – double array**

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

4: **s(npde) – double array**

s(i) must be set to the value of $S_i\left(x,t,U,V,\dot{V}\right)$, 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 d03pl returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

5: **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 d03pl.

Its specification is:

```
[flux, ires] = numflx(npde, t, x, ncode, v, 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: **ncode – int32 scalar**

The number of coupled ODEs in the system.

5: **v**$(*)$ **– double array**

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

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

6: **uleft**(**npde**) **– double array**

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

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

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

8: **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 d03pl 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 d03pl returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

6: **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 (8) and (9).

Its specification is:

```
[g, ires] = bndary(npde, npts, t, x, u, ncode, v, vdot, 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. $\mathbf{x}(1)$ corresponds to the left-hand boundary, a, and $\mathbf{x}(\mathbf{npts})$ corresponds to the right-hand boundary, b.

5: **u(npde,npts) – double array**

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

Note: if banded matrix algebra is to be used then the functions G_i^L and G_i^R may depend on the value of $U_i(x,t)$ at the boundary point and the two adjacent points only.

6: **ncode – int32 scalar**

The number of coupled ODEs in the system.

7: **v(*) – double array**

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

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

8: **vdot(*) – double array**

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

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

Note: $\dot{V}_i(t)$, for $i = 1,2,\ldots,\mathbf{ncode}$, may only appear linearly in G_j^L and G_j^R, for $j = 1,2,\ldots,\mathbf{npde}$.

9: **ibnd – int32 scalar**

Specifies which boundary conditions are to be evaluated.

$\mathbf{ibnd} = 0$

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

$\mathbf{ibnd} \neq 0$

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

10: **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:

$\mathbf{ires} = 2$

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

$\mathbf{ires} = 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set $\mathbf{ires} = 3$ when a physically meaningless input or output value has been generated. If you consecutively set $\mathbf{ires} = 3$, then d03pl returns to the calling (sub)program with the error indicator set to $\mathbf{ifail} = 4$.

Output Parameters

1: **g(npde) – double array**

g(i) must contain the ith component of either G_i^L or G_i^R in (8) and (9), 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 d03pl returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

7: **u(neqn) – double array**

The initial values of the dependent variables defined as follows:

u(**npde** $\times (j-1) + i$) contain $U_i(x_j, t_0)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**, and **u**(**npts** $\times$ **npde** $+ k$) contain $V_k(t_0)$, for $k = 1, 2, \ldots,$ **ncode**.

8: **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**).

9: **ncode – int32 scalar**

The number of coupled ODE components.

Constraint: **ncode** ≥ 0.

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

odedef must evaluate the functions R, 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:

```
[r, 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 $i = 1, 2, \ldots,$ **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 $i = 1, 2, \ldots,$ **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 $i = 1, 2, \ldots,$ **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 $\max(1,$ **nxi**$)$

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

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

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

11: **ires – int32 scalar**

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

ires $= 1$

Equation (10) must be used.

ires $= -1$

Equation (11) 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 d03pl returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **r**$(*)$ – **double array**

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

r(i) must contain the ith component of R, for $i = 1, 2, \ldots,$ **ncode**, where R is defined as

$$R = L - M\dot{V} - NU_t^*, \tag{10}$$

or

$$R = -M\dot{V} - NU_t^*. \tag{11}$$

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

2: **ires** – **int32 scalar**

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

ires $= 1$

Equation (10) must be used.

ires $= -1$

Equation (11) 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 d03pl returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

11: **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.

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

12: **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: **rtol**(i) ≥ 0 for all relevant i.

13: **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: **atol**(i) ≥ 0 for all relevant i.

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

14: **itol** – **int32 scalar**

A value to indicate the form of the local error test. If e_i is the estimated local error for **u**(i), for $i = 1, 2, \ldots, \mathbf{neqn}$, and $\| \quad \|$ denotes the norm, then the error test to be satisfied is $\|e_i\| < 1.0$. **itol** indicates to d03pl whether to interpret either or both of **rtol** and **atol** as a vector or scalar in the formation of the weights w_i used in the calculation of the norm (see the description of **norm_p**):

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	$\mathbf{rtol}(i) \times \lvert\mathbf{u}(i)\rvert + \mathbf{atol}(i)$

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

15: **norm_p** – **string**

The type of norm to be used.

norm_p = '1'

Averaged L_1 norm.

norm_p = '2'

Averaged L_2 norm.

If U_{norm} denotes the norm of the vector **u** of length **neqn**, then for the averaged L_1 norm

$$U_{\mathrm{norm}} = \frac{1}{\mathbf{neqn}} \sum_{i=1}^{\mathbf{neqn}} \mathbf{u}(i)/w_i,$$

and for the averaged L_2 norm

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

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

Constraint: **norm_p** = '1' or '2'.

16: **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 (**ncode** = 0). Also, the banded option should not be used if the boundary conditions involve solution components at points other than the boundary and the immediately adjacent two points.

17: **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 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 $j = 1, 2, \ldots, \mathbf{npde}$ 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**$(5) \leq 0.99$. 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 the 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 the 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 matrix is found to be numerically singular (see **algopt**(29)). The default value is **algopt**(30) $= 0.0001$.

18: **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.

19: **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 BDF method last used in the time integration, if applicable. When the Theta method is used **isave**(4) contains no useful information.

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.

20: **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.

21: **itrace – int32 scalar**

The level of trace information required from d03pl 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.

22: **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 d03pl.

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: **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** $\geq (3 \times MLU + 1) \times$ **neqn** $+ NWKRES + LENODE$.

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

Where

$MLU =$ the lower or upper half bandwidths, and
$MLU = 3 \times$ **npde** $- 1$, for PDE problems only, and
$MLU =$ **neqn** $- 1$, for coupled PDE/ODE problems.

$NWKRES =$ **npde** $\times (2 \times$ **npts** $+ 6 \times$ **nxi** $+ 3 \times$ **npde** $+ 26) +$ **nxi** $+$
ncode $+ 7 \times$ **npts** $+ 2$, when **ncode** > 0 and **nxi** > 0, and
$NWKRES =$ **npde** $\times (2 \times$ **npts** $+ 3 \times$ **npde** $+ 32) +$ **ncode** $+ 7 \times$ **npts** $+ 3$,
when **ncode** > 0 and **nxi** $= 0$, and
$NWKRES =$ **npde** $\times (2 \times$ **npts** $+ 3 \times$ **npde** $+ 32) + 7 \times$ **npts** $+ 4$,
when **ncode** $= 0$.

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

Note: when **laopt** = 'S', 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** $= 15$.

5: **lisave – int32 scalar**

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

Its size depends on the type of matrix algebra selected:

if **laopt** = 'F', **lisave** ≥ 24;
if **laopt** = 'B', **lisave** $\geq$ **neqn** $+ 24$;
if **laopt** = 'S', **lisave** $\geq 25 \times$ **neqn** $+ 24$.

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**

The computed solution $U_i(x_j, t)$, for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **npts**, and $V_k(t)$, for $k = 1, 2, \ldots,$ **ncode**, all evaluated at $t =$ **ts**.

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.

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 BDF method last used in the time integration, if applicable. When the Theta method is used **isave**(4) contains no useful information.

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.

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** $\neq 1, 2, 3, 4$ or 5,
or at least one of the coupling points defined in array **xi** is outside the interval $[\mathbf{x}(1), \mathbf{x}(\mathbf{npts})]$,
or the coupling points are not in strictly increasing order,
or **npts** < 3,
or **npde** < 1,
or **laopt** $\neq$ 'F' , 'B' or 'S',
or **itol** $\neq 1, 2, 3$ or 4,
or **ind** $\neq 0$ or 1,
or mesh points $\mathbf{x}(i)$ badly ordered,
or **lrsave** or **lisave** are too small,
or **ncode** and **nxi** are incorrectly defined,
or **ind** $= 1$ on initial entry to d03pl,
or **neqn** $\neq$ **npde** $\times$ **npts** $+$ **ncode**,
or an element of **rtol** or **atol** < 0.0,
or corresponding elements of **rtol** and **atol** are both 0.0,
or **norm_p** $\neq 1$ or 2.

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 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**, **bndary** or **odedef**, 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**, **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**, **numflx**, **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 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 **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 when **itrace** ≥ 1). 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) has been taken.

ifail $= 13$

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

ifail $= 14$

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.

ifail $= 15$

When using the sparse option, the value of **lisave** or **lrsave** was not sufficient (more detailed information may be directed to the current error message unit).

7 Accuracy

d03pl 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

d03pl 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 **algopt**(13). 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. For a given system and a fixed accuracy it is approximately proportional to **neqn**.

9 Example

d03pl_bndary.m

```
function [g, ires] = bndary(npde, npts, t, x, u, ncode, v, vdot, ibnd,
ires)
  g = zeros(npde, 1);
  if (ibnd == 0)
     ue = d03pl_exact(t,npde,x(1),1);
     g(1) = u(1,1) + u(2,1) - ue(1,1) - ue(2,1);
     dudx = (u(1,2)-u(2,2)-u(1,1)+u(2,1))/(x(2)-x(1));
     g(2) = vdot(1) - dudx;
  else
     ue = d03pl_exact(t,npde,x(npts),1);
     g(1) = u(1,npts) - u(2,npts) - ue(1,1) + ue(2,1);
        dudx  =  (u(1,npts)+u(2,npts)-u(1,npts-1)-u(2,npts-1))/(x(npts)-
x(npts-1));
     g(2) = vdot(2) + 3*dudx;
  end
```

d03pl_exact.m

```
function [u] = exact(t,npde,x,npts)
  u = zeros(npde,npts);
  pi2 = 2*pi;
  for i = 1:npts
   f = exp(pi*(x(i)-3*t))*sin(pi2*(x(i)-3*t));
   g = exp(-pi2*(x(i)+t))*cos(pi2*(x(i)+t));
   u(1,i) = f + g;
   u(2,i) = f - g;
  end
```

d03pl_numflx.m

```
function  [flux,  ires]  =  numflx(npde,  t,  x,  ncode,  v,  uleft,  uright,
ires)
  flux = zeros(npde, 1);
  flux(1) = 0.5*(3.0*uleft(1)-uright(1)+3.0*uleft(2)+uright(2));
  flux(2) = 0.5*(3.0*uleft(1)+uright(1)+3.0*uleft(2)-uright(2));
```

d03pl_odedef.m

```
function  [r,  ires]  =  odedef(npde,  t,  ncode,  v,  vdot,  nxi,  xi,  ucp,
ucpx, ucpt, ires)
  r = zeros(ncode, 1);
  if (ires ~= -1)
    r(1) = v(1) - ucp(1,1) + ucp(2,1);
    r(2) = v(2) - ucp(1,2) - ucp(2,2);
  end
```

```
d03pl_pdedef.m
```

```
function [p, c, d, s, ires] = pdedef(npde, t, x, u, ux, ncode, v, vdot,
ires)
  p = zeros(npde, npde);
  c = zeros(npde, 1);
  d = zeros(npde, 1);
  s = zeros(npde, 1);
  for i = 1:npde
     c(i) = 1;
     for j = 1:npde
        if (i == j)
           p(i,j) = 1;
        end
     end
  end
```

```
npde = int32(2);
npts = int32(141);
ncode = int32(2);
neqn = npde*npts+ncode;
ts = 0;
tout = 0.5;
x = zeros(npts,1);
xi = [0; 1];
rtol = [0.00025];
atol = [1e-05];
itol = int32(1);
norm_p = '1';
laopt = 'S';
algopt = [1;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0;
     0.1;
     1.1];
rsave = zeros(11000, 1);
isave = zeros(15700, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
```

```
% Initialise mesh
for i=1:npts
  x(i) = double(i-1)/double(npts-1);
end

u = d03pl_exact(ts, npde, x, npts);
u = reshape(u, [282,1]);
u(neqn-1) = u(1) - u(2);
u(neqn) = u(neqn-2) + u(neqn - 3);

[tsOut, uOut, rsaveOut, isaveOut, indOut, ifail] = ...
    d03pl(npde, ts, tout, 'd03pl_pdedef', 'd03pl_numflx', 'd03pl_bndary',
...
     u, x, ncode, 'd03pl_odedef', xi, rtol, atol, itol, norm_p, laopt,
algopt, ...
    rsave, isave, itask, itrace, ind);

nop = 0;
xout = zeros(8,1);
for i=1:20:npts
  nop = nop+1;
  xout(nop) = x(i);
end

uout=zeros(npde, 8);
for i = 1:nop
  ii = 1 + 20*(i-1);
  jj = npde*(ii-1);
  uout(1, i) = uOut(jj+1);
  uout(2, i) = uOut(jj+2);
end

ue = d03pl_exact(tout, npde, xout, nop);

output = zeros(nop, 5);
fprintf('\n         X         Approx U1   Exact U1    Approx U2   Exact
U2\n');
for i=1:nop
  fprintf('   %9.4f   %9.4f   %9.4f   %9.4f   %9.4f\n', ...
          xout(i), uout(1,i), ue(1,i), uout(2,i), ue(2,i));
  output(i,:) = [xout(i), uout(1,i), ue(1,i), uout(2,i), ue(2,i)];
end
```

```
X        Approx U1   Exact U1    Approx U2   Exact U2
0.0000     -0.0432     -0.0432      0.0432      0.0432
      0.1429     -0.0220     -0.0220     -0.0001     -0.0000
      0.2857     -0.0200     -0.0199     -0.0231     -0.0231
      0.4286     -0.0123     -0.0123     -0.0176     -0.0176
      0.5714      0.0247      0.0245      0.0227      0.0224
      0.7143      0.0833      0.0827      0.0831      0.0825
      0.8571      0.1041      0.1036      0.1043      0.1039
      1.0000     -0.0007     -0.0001     -0.0006      0.0001
```