NAG Library Routine Document

D03PLF

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

1 Purpose

D03PLF 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 Specification

```
SUBROUTINE D03PLF (NPDE, TS, TOUT, PDEDEF, NUMFLX, BNDARY, U, NPTS, X,      &
                   NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL, ITOL, NORM,    &
                   LAOPT, ALGOPT, RSAVE, LRSAVE, ISAVE, LISAVE, ITASK,      &
                   ITRACE, IND, IFAIL)
INTEGER            NPDE, NPTS, NCODE, NXI, NEQN, ITOL, LRSAVE,              &
                   ISAVE(LISAVE), LISAVE, ITASK, ITRACE, IND, IFAIL
REAL (KIND=nag_wp) TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*), ATOL(*),     &
                   ALGOPT(30), RSAVE(LRSAVE)
CHARACTER(1)       NORM, LAOPT
EXTERNAL           PDEDEF, NUMFLX, BNDARY, ODEDEF
```

3 Description

D03PLF integrates the system of convection-diffusion equations in conservative form:

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

or the hyperbolic convection-only system:

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

for $i=1,2,\dots,\mathrm{NPDE}$, $\quad a\leq x\leq b$, $\quad t\geq t_0$, where the vector U is the set of PDE solution values

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

The optional coupled ODEs are of the general form

$$R_i\big(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*\big)=0, \quad i=1,2,\dots,\mathrm{NCODE}, \tag{3}$$

where the vector V is the set of ODE solution values

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

$\dot{V}$ 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 $R = [R_1, \ldots, R_{\text{NCODE}}]^{\text{T}}$, L is a vector of length NCODE, M is an NCODE by NCODE matrix, N is an NCODE by $\left(n_\xi \times \text{NPDE}\right)$ 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 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 $b = x_{\text{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\text{NPTS}}$. 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 D03PCF/D03PCA, D03PHF/D03PHA and D03PPF/D03PPA, 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 $j = 2, 3, \ldots, \text{NPTS}$. The left and right values are calculated by D03PLF 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

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

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

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

An example is given in Section 10 and in the D03PFF 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 PDEDEF. The numerical flux $\hat{F}_i$ must be supplied in a separate NUMFLX. For problems in the form (2), the actual argument D03PLP may be used for PDEDEF. D03PLP is included in the NAG 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 10 and in the D03PFF documentation.

The boundary conditions must be specified in BNDARY in the form

$$G_i^L\big(x,t,U,V,\dot{V}\big) = 0 \quad \text{at } x = a, \quad i = 1,2,\ldots,\text{NPDE}, \tag{8}$$

at the left-hand boundary, and

$$G_i^R\big(x,t,U,V,\dot{V}\big) = 0 \quad \text{at } x = b, \quad i = 1,2,\ldots,\text{NPDE}, \tag{9}$$

at the right-hand boundary.

Note that spatial derivatives at the boundary are not passed explicitly to 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 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 $j = 1,2,\ldots,\text{NCODE}$, may only appear **linearly** in the functions S_i, for $i = 1,2,\ldots,\text{NPDE}$, 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) $t_0 < t_{\text{out}}$, 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 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 $x_1, x_2, \ldots, x_{\text{NPTS}}$;

(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 $\text{NPDE} \times \text{NPTS} + \text{NCODE}$ 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

1: NPDE – INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: TS – REAL (KIND=nag_wp) *Input/Output*

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

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

Constraint: TS < TOUT.

3: TOUT – REAL (KIND=nag_wp) *Input*

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

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

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 D03PLF. The actual argument D03PLP may be used for PDEDEF for problems in the form (2). (D03PLP is included in the NAG Library.)

The specification of PDEDEF is:

```
SUBROUTINE PDEDEF (NPDE, T, X, U, UX, NCODE, V, VDOT, P, C, D, S,    &
                   IRES)

INTEGER            NPDE, NCODE, IRES
REAL (KIND=nag_wp) T, X, U(NPDE), UX(NPDE), V(NCODE),                &
                   VDOT(NCODE), P(NPDE,NPDE), C(NPDE), D(NPDE),      &
                   S(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

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

3: X – REAL (KIND=nag_wp) *Input*

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

4: U(NPDE) – REAL (KIND=nag_wp) array *Input*

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

5: UX(NPDE) – REAL (KIND=nag_wp) array *Input*

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

6: NCODE – INTEGER *Input*

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

7: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if $\mathrm{NCODE} > 0$, $\mathrm{V}(i)$ contains the value of the component $V_i(t)$, for $i = 1,2,\ldots,\mathrm{NCODE}$.

8: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if $\mathrm{NCODE} > 0$, $\mathrm{VDOT}(i)$ contains the value of component $\dot{V}_i(t)$, for $i = 1,2,\ldots,\mathrm{NCODE}$.

Note: $\dot{V}_i(t)$, for $i = 1,2,\ldots,\mathrm{NCODE}$, may only appear linearly in S_j, for $j = 1,2,\ldots,\mathrm{NPDE}$.

9: P(NPDE, NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{P}(i,j)$ must be set to the value of $P_{i,j}(x,t,U,V)$, for $i = 1,2,\ldots,\mathrm{NPDE}$ and $j = 1,2,\ldots,\mathrm{NPDE}$.

10: C(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{C}(i)$ must be set to the value of $C_i(x,t,U,V)$, for $i = 1,2,\ldots,\mathrm{NPDE}$.

11: D(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{D}(i)$ must be set to the value of $D_i(x,t,U,U_x,V)$, for $i = 1,2,\ldots,\mathrm{NPDE}$.

12: S(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{S}(i)$ must be set to the value of $S_i\big(x,t,U,V,\dot{V}\big)$, for $i = 1,2,\ldots,\mathrm{NPDE}$.

13: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

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

$\mathrm{IRES} = 2$
: Indicates to the integrator that control should be passed back immediately to the calling subroutine with the error indicator set to $\mathrm{IFAIL} = 6$.

$\mathrm{IRES} = 3$
: Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set $\mathrm{IRES} = 3$ when a physically meaningless input or output value has been generated. If you consecutively set $\mathrm{IRES} = 3$, then D03PLF returns to the calling subroutine with the error indicator set to $\mathrm{IFAIL} = 4$.

PDEDEF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

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 D03PLF.

The specification of NUMFLX is:

```
SUBROUTINE NUMFLX (NPDE, T, X, NCODE, V, ULEFT, URIGHT, FLUX,        &
                   IRES)
INTEGER            NPDE, NCODE, IRES
REAL (KIND=nag_wp) T, X, V(NCODE), ULEFT(NPDE), URIGHT(NPDE),        &
                   FLUX(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

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

3: X – REAL (KIND=nag_wp) *Input*

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

4: NCODE – INTEGER *Input*

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

5: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if $\text{NCODE} > 0$, $\text{V}(i)$ contains the value of the component $V_i(t)$, for $i = 1, 2, \ldots, \text{NCODE}$.

6: ULEFT(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: $\text{ULEFT}(i)$ contains the *left* value of the component $U_i(x)$, for $i = 1, 2, \ldots, \text{NPDE}$.

7: URIGHT(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: $\text{URIGHT}(i)$ contains the *right* value of the component $U_i(x)$, for $i = 1, 2, \ldots, \text{NPDE}$.

8: FLUX(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: $\text{FLUX}(i)$ must be set to the numerical flux $\hat{F}_i$, for $i = 1, 2, \ldots, \text{NPDE}$.

9: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

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

$\text{IRES} = 2$

Indicates to the integrator that control should be passed back immediately to the calling subroutine with the error indicator set to $\text{IFAIL} = 6$.

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 D03PLF returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

NUMFLX must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

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).

The specification of BNDARY is:

```
SUBROUTINE BNDARY (NPDE, NPTS, T, X, U, NCODE, V, VDOT, IBND, G,     &
                   IRES)
INTEGER            NPDE, NPTS, NCODE, IBND, IRES
REAL (KIND=nag_wp) T, X(NPTS), U(NPDE,NPTS), V(NCODE),               &
                   VDOT(NCODE), G(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: NPTS – INTEGER *Input*

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

3: T – REAL (KIND=nag_wp) *Input*

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

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: 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, NPTS) – REAL (KIND=nag_wp) array *Input*

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

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 – INTEGER *Input*

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

7: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, V(i) contains the value of the component $V_i(t)$, for $i = 1, 2, \ldots, \mathrm{NCODE}$.

8: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots, \mathrm{NCODE}$.

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

9: IBND – INTEGER *Input*

On entry: 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 $x = b$.

10: G(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: 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 $i = 1, 2, \ldots, \text{NPDE}$.

11: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

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

IRES $= 2$
Indicates to the integrator that control should be passed back immediately to the calling subroutine 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 D03PLF returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

BNDARY must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the initial values of the dependent variables defined as follows:

U$(\text{NPDE} \times (j-1) + i)$ contain $U_i(x_j, t_0)$, for $i = 1, 2, \ldots, \text{NPDE}$ and $j = 1, 2, \ldots, \text{NPTS}$, and

U$(\text{NPTS} \times \text{NPDE} + k)$ contain $V_k(t_0)$, for $k = 1, 2, \ldots, \text{NCODE}$.

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

8: NPTS – INTEGER *Input*

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

Constraint: NPTS ≥ 3.

9: X(NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: 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).

10: NCODE – INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

11: ODEDEF – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

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 dummy routine D03PEK. (D03PEK is included in the NAG Library.)

The specification of ODEDEF is:

```
SUBROUTINE ODEDEF (NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,      &
                   UCPT, R, IRES)

INTEGER            NPDE, NCODE, NXI, IRES
REAL (KIND=nag_wp) T, V(NCODE), VDOT(NCODE), XI(NXI),                &
                   UCP(NPDE,*), UCPX(NPDE,*), UCPT(NPDE,*),          &
                   R(NCODE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

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

3: NCODE – INTEGER *Input*

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

4: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, V(i) contains the value of the component $V_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

5: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

6: NXI – INTEGER *Input*

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

7: XI(NXI) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, XI(i) contains the ODE/PDE coupling point, ξ_i, for $i = 1, 2, \ldots,$ NXI.

8: UCP(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, 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, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, 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, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, UCPT(i,j) contains the value of $\frac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

11: R(NCODE) – REAL (KIND=nag_wp) array *Output*

On exit: 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.

12: IRES – INTEGER *Input/Output*

On entry: 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.

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

IRES $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)routine 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 D03PLF returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

ODEDEF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: NXI – INTEGER *Input*

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

Constraints:

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

13: XI($*$) – REAL (KIND=nag_wp) array *Input*

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

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

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

14: NEQN – INTEGER *Input*

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

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

15: RTOL($*$) – REAL (KIND=nag_wp) array *Input*

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

On entry: the relative local error tolerance.

Constraint: $\mathrm{RTOL}(i) \geq 0.0$ for all relevant i.

16: ATOL($*$) – REAL (KIND=nag_wp) array *Input*

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

On entry: the absolute local error tolerance.

Constraint: $\mathrm{ATOL}(i) \geq 0.0$ for all relevant i.

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

17: ITOL – INTEGER *Input*

On entry: a value to indicate the form of the local error test. If e_i is the estimated local error for $\mathrm{U}(i)$, for $i = 1, 2, \ldots, \mathrm{NEQN}$, and $\| \quad \|$ denotes the norm, then the error test to be satisfied is $\|e_i\| < 1.0$. ITOL indicates to D03PLF 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):

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

Constraint: $1 \leq \mathrm{ITOL} \leq 4$.

18: NORM – CHARACTER(1) *Input*

On entry: the type of norm to be used.

NORM = '1'
Averaged L_1 norm.

NORM = '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

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

and for the averaged L_2 norm

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

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

Constraint: NORM = '1' or '2'.

19: LAOPT – CHARACTER(1) *Input*

On entry: the type of matrix algebra required.

LAOPT = 'F'
: Full matrix methods to be used.

LAOPT = 'B'
: Banded matrix methods to be used.

LAOPT = 'S'
: Sparse matrix methods to be used.

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

Note: you are recommended to use the banded option when no coupled ODEs are present (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.

20: ALGOPT(30) – REAL (KIND=nag_wp) array *Input*

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

ALGOPT(1)
: Selects the ODE integration method to be used. If ALGOPT$(1) = 1.0$, a BDF method is used and if ALGOPT$(1) = 2.0$, a Theta method is used. The default 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$(5) \leq 0.99$. 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 routines regard the Jacobian matrix as numerically singular, then increasing ALGOPT(29) towards 1.0 may help, but at the cost of increased fill-in. The default value is ALGOPT(29) $= 0.1$.

ALGOPT(30)
Used as 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) $= 0.0001$.

21: RSAVE(LRSAVE) – REAL (KIND=nag_wp) array *Communication Array*

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

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

22: LRSAVE – INTEGER *Input*

On entry: the dimension of the array RSAVE as declared in the (sub)program from which D03PLF is called. 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', $\text{LRSAVE} \geq 4 \times \text{NEQN} + 11 \times \text{NEQN}/2 + 1 + nwkres + lenode$.

Where

mlu is the lower or upper half bandwidths such that
$mlu = 3 \times \text{NPDE} - 1$, for PDE problems only (no coupled ODEs); or
$mlu = \text{NEQN} - 1$, for coupled PDE/ODE problems.

$$nwkres = \begin{cases} \text{NPDE} \times (2 \times \text{NPTS} + 6 \times \text{NXI} + 3 \times \text{NPDE} + 26) + \text{NXI} + \text{NCODE} + 7 \times \text{NPTS} + 2, & \text{when NCODE} > 0 \text{ and NXI} > 0; \\ \text{NPDE} \times (2 \times \text{NPTS} + 3 \times \text{NPDE} + 32) + \text{NCODE} + 7 \times \text{NPTS} + 3, & \text{when NCODE} > 0 \text{ and NXI} = 0; \\ \text{NPDE} \times (2 \times \text{NPTS} + 3 \times \text{NPDE} + 32) + 7 \times \text{NPTS} + 4, & \text{when NCODE} = 0. \end{cases}$$

$$lenode = \begin{cases} (6 + \text{int}(\text{ALGOPT}(2))) \times \text{NEQN} + 50, & \text{when the BDF method is used; or} \\ 9 \times \text{NEQN} + 50, & \text{when the Theta method is used.} \end{cases}$$

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 routine returns with IFAIL $= 15$.

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

If IND $= 0$, ISAVE need not be set.

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

ISAVE(1)
Contains the number of steps taken in time.

ISAVE(2)
Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves evaluating the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

ISAVE(3)
Contains the number of Jacobian evaluations performed by the time integrator.

ISAVE(4)
Contains the order of the 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.

24: LISAVE – INTEGER *Input*

On entry: the dimension of the array ISAVE as declared in the (sub)program from which D03PLF is called. Its size depends on the type of matrix algebra selected:

if LAOPT = 'F', $\text{LISAVE} \geq 24$;

if LAOPT = 'B', $\text{LISAVE} \geq \text{NEQN} + 24$;

if LAOPT = 'S', $\text{LISAVE} \geq 25 \times \text{NEQN} + 24$.

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 routine returns with IFAIL $= 15$.

25: ITASK – INTEGER *Input*

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

ITASK $= 1$
Normal computation of output values U at $t = \text{TOUT}$ (by overshooting and interpolating).

ITASK $= 2$
Take one step in the time direction and return.

ITASK $= 3$
Stop at first internal integration point at or beyond $t = \text{TOUT}$.

ITASK $= 4$
Normal computation of output values U at $t = \text{TOUT}$ but without overshooting $t = t_{\text{crit}}$ where t_{crit} is described under the parameter ALGOPT.

ITASK $= 5$
Take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: ITASK $= 1$, 2, 3, 4 or 5.

26: ITRACE – INTEGER *Input*

On entry: the level of trace information required from D03PLF 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 X04AAF).

ITRACE > 0
Output from the underlying ODE solver is printed on the current advisory message unit (see X04ABF). This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

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.

27: IND – INTEGER *Input/Output*

On entry: indicates whether this is a continuation call or a new integration.

IND $= 0$
Starts or restarts the integration in time.

IND $= 1$
Continues the integration after an earlier exit from the routine. In this case, only the parameters TOUT and IFAIL should be reset between calls to D03PLF.

Constraint: IND $= 0$ or 1.

On exit: IND $= 1$.

28: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this parameter, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

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

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

IFAIL $= 1$

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 $[\mathrm{X}(1), \mathrm{X}(\mathrm{NPTS})]$,
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 $\mathrm{X}(i)$ are badly ordered,
or LRSAVE or LISAVE are too small,
or NCODE and NXI are incorrectly defined,
or IND $= 1$ on initial entry to D03PLF,
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 $\neq$ 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 routine was unable to initialize the derivative of the ODE system. This could be due to the fact that IRES was repeatedly set to 3 in one of 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 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 routine is unable to start the integration in time.

IFAIL $= 8$

In either, PDEDEF, NUMFLX, BNDARY or ODEDEF, IRES was set to an invalid value.

IFAIL $= 9$ (D02NNF)

A serious error has occurred in an internal call to the specified routine. Check the 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).

IFAIL $= -99$

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

D03PLF 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 Parallelism and Performance

D03PLF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D03PLF makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

D03PLF 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 routine 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.

10 Example

For this routine two examples are presented, with a main program and two example problems given in Example 1 (EX1) and Example 2 (EX2).

Example 1 (EX1)

This example is a simple first-order system with coupled ODEs arising from the use of the characteristic equations for the numerical boundary conditions.

The PDEs are

$$\frac{\partial U_1}{\partial t} + \frac{\partial U_1}{\partial x} + 2\frac{\partial U_2}{\partial x} = 0,$$

$$\frac{\partial U_2}{\partial t} + 2\frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

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

The PDEs have an exact solution given by

$$U_1(x,t) = f(x-3t) + g(x+t), \quad U_2(x,t) = f(x-3t) - g(x+t),$$

where $f(z) = \exp(\pi z)\sin(2\pi z)$, $g(z) = \exp(-2\pi z)\cos(2\pi z)$.

The initial conditions are given by the exact solution.

The characteristic variables are $W_1 = U_1 - U_2$ and $W_2 = U_1 + U_2$, corresponding to the characteristics given by $dx/dt = -1$ and $dx/dt = 3$ respectively. Hence we require a physical boundary condition for W_2 at the left-hand boundary and for W_1 at the right-hand boundary (corresponding to the incoming characteristics), and a numerical boundary condition for W_1 at the left-hand boundary and for W_2 at the right-hand boundary (outgoing characteristics).

The physical boundary conditions are obtained from the exact solution, and the numerical boundary conditions are supplied in the form of the characteristic equations for the outgoing characteristics, that is

$$\frac{\partial W_1}{\partial t} - \frac{\partial W_1}{\partial x} = 0$$

at the left-hand boundary, and

$$\frac{\partial W_2}{\partial t} + 3\frac{\partial W_2}{\partial x} = 0$$

at the right-hand boundary.

In order to specify these boundary conditions, two ODE variables V_1 and V_2 are introduced, defined by

$$\begin{aligned} V_1(t) &= W_1(0,t) = U_1(0,t) - U_2(0,t), \\ V_2(t) &= W_2(1,t) = U_1(1,t) + U_2(1,t). \end{aligned}$$

The coupling points are therefore at $x = 0$ and $x = 1$.

The numerical boundary conditions are now

$$\dot{V}_1 - \frac{\partial W_1}{\partial x} = 0$$

at the left-hand boundary, and

$$\dot{V}_2 + 3\frac{\partial W_2}{\partial x} = 0$$

at the right-hand boundary.

The spatial derivatives are evaluated at the appropriate boundary points in BNDARY using one-sided differences (into the domain and therefore consistent with the characteristic directions).

The numerical flux is calculated using Roe's approximate Riemann solver (see Section 3 for details), giving

$$\hat{F} = \tfrac{1}{2}\begin{bmatrix} 3U_{1L} - U_{1R} + 3U_{2L} + U_{2R} \\ 3U_{1L} + U_{1R} + 3U_{2L} - U_{2R} \end{bmatrix}.$$

Example 2 (EX2)

This example is the standard shock-tube test problem proposed by Sod (1978) for the Euler equations of gas dynamics. The problem models the flow of a gas in a long tube following the sudden breakdown of a diaphragm separating two initial gas states at different pressures and densities. There is an exact solution to this problem which is not included explicitly as the calculation is quite lengthy. The PDEs are

$$\begin{aligned} \frac{\partial \rho}{\partial t}+\frac{\partial m}{\partial x} &= 0, \\ \frac{\partial m}{\partial t}+\frac{\partial}{\partial x}\Big(\tfrac{m^2}{\rho}+(\gamma-1)\Big(e-\tfrac{m^2}{2\rho}\Big)\Big) &= 0, \\ \frac{\partial e}{\partial t}+\frac{\partial}{\partial x}\Big(\tfrac{me}{\rho}+\tfrac{m}{\rho}(\gamma-1)\Big(e-\tfrac{m^2}{2\rho}\Big)\Big) &= 0, \end{aligned}$$

where ρ is the density; m is the momentum, such that $m=\rho u$, where u is the velocity; e is the specific energy; and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p=(\gamma-1)\left(e-\frac{\rho u^2}{2}\right).$$

The solution domain is $0 \le x \le 1$ for $0 < t \le 0.2$, with the initial discontinuity at $x = 0.5$, and initial conditions

$$\begin{array}{llll} \rho(x,0)=1, & m(x,0)=0, & e(x,0)=2.5, & \text{for } x<0.5, \\ \rho(x,0)=0.125, & m(x,0)=0, & e(x,0)=0.25, & \text{for } x>0.5. \end{array}$$

The solution is uniform and constant at both boundaries for the spatial domain and time of integration stated, and hence the physical and numerical boundary conditions are indistinguishable and are both given by the initial conditions above. The evaluation of the numerical flux for the Euler equations is not trivial; the Roe algorithm given in Section 3 cannot be used directly as the Jacobian is nonlinear. However, an algorithm is available using the parameter-vector method (see Roe (1981)), and this is provided in the utility routine D03PUF. An alternative Approxiate Riemann Solver using Osher's scheme is provided in D03PVF. Either D03PUF or D03PVF can be called from NUMFLX.

10.1 Program Text

```
!   D03PLF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module d03plfe_mod

!     D03PLF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                              :: bndry1, bndry2, exact, nmflx1,   &
                                             nmflx2, odedef, pdedef, uvinit
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: el0 = 2.5_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: er0 = 0.25_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: gamma = 1.4_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: rl0 = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: rr0 = 0.125_nag_wp
      Integer, Parameter, Public          :: itrace = 0, ncode1 = 2,          &
                                             ncode2 = 0, nin = 5, nout = 6,   &
                                             npde1 = 2, npde2 = 3, nxi1 = 2,  &
                                             nxi2 = 0
    Contains
      Subroutine exact(t,u,npde,x,npts)
!       Exact solution (for comparison and b.c. purposes)

!       .. Use Statements ..
        Use nag_library, Only: x01aaf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
        Integer, Intent (In)                :: npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: u(npde,npts)
        Real (Kind=nag_wp), Intent (In)     :: x(*)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: f, g, pi, pi2, x1, x3
        Integer                             :: i
!       .. Intrinsic Procedures ..
        Intrinsic                           :: cos, exp, sin
!       .. Executable Statements ..
        f = 0.0_nag_wp
        pi = x01aaf(f)
        pi2 = 2.0_nag_wp*pi
        Do i = 1, npts
          x1 = x(i) + t
          x3 = x(i) - 3.0_nag_wp*t
          f = exp(pi*x3)*sin(pi2*x3)
          g = exp(-pi2*x1)*cos(pi2*x1)
          u(1,i) = f + g
          u(2,i) = f - g
        End Do
        Return
      End Subroutine exact
      Subroutine pdedef(npde,t,x,u,ux,ncode,v,vdot,p,c,d,s,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t, x
        Integer, Intent (Inout)             :: ires
        Integer, Intent (In)                :: ncode, npde
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: c(npde), d(npde),              &
                                               p(npde,npde), s(npde)
        Real (Kind=nag_wp), Intent (In)     :: u(npde), ux(npde), v(ncode),   &
                                               vdot(ncode)
!       .. Local Scalars ..
        Integer                             :: i
!       .. Executable Statements ..
        c(1:npde) = 1.0_nag_wp
        d(1:npde) = 0.0_nag_wp
        s(1:npde) = 0.0_nag_wp
        p(1:npde,1:npde) = 0.0_nag_wp
        Do i = 1, npde
          p(i,i) = 1.0_nag_wp
        End Do
        Return
      End Subroutine pdedef
      Subroutine bndry1(npde,npts,t,x,u,ncode,v,vdot,ibnd,g,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: t
        Integer, Intent (In)                :: ibnd, ncode, npde, npts
        Integer, Intent (Inout)             :: ires
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: g(npde)
        Real (Kind=nag_wp), Intent (In)     :: u(npde,npts), v(ncode),        &
                                               vdot(ncode), x(npts)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                  :: dudx
        Integer                             :: i
!       .. Local Arrays ..
        Real (Kind=nag_wp)                  :: ue(2,1)
!       .. Executable Statements ..
        If (ibnd==0) Then
          i = 1
          Call exact(t,ue,npde,x(i),1)
          g(1) = u(1,i) + u(2,i) - ue(1,1) - ue(2,1)
          dudx = (u(1,i+1)-u(2,i+1)-u(1,i)+u(2,i))/(x(i+1)-x(i))
          g(2) = vdot(1) - dudx
        Else
          i = npts
          Call exact(t,ue,npde,x(i),1)
          g(1) = u(1,i) - u(2,i) - ue(1,1) + ue(2,1)
          dudx = (u(1,i)+u(2,i)-u(1,i-1)-u(2,i-1))/(x(i)-x(i-1))
          g(2) = vdot(2) + 3.0_nag_wp*dudx
        End If
        Return
      End Subroutine bndry1
      Subroutine nmflx1(npde,t,x,ncode,v,uleft,uright,flux,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t, x
        Integer, Intent (Inout)              :: ires
        Integer, Intent (In)                 :: ncode, npde
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: flux(npde)
        Real (Kind=nag_wp), Intent (In)      :: uleft(npde), uright(npde),     &
                                                v(ncode)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: tmpl, tmpr
!       .. Executable Statements ..
        tmpl = 3.0_nag_wp*(uleft(1)+uleft(2))
        tmpr = uright(1) - uright(2)
        flux(1) = 0.5_nag_wp*(tmpl-tmpr)
        flux(2) = 0.5_nag_wp*(tmpl+tmpr)
        Return
      End Subroutine nmflx1
      Subroutine odedef(npde,t,ncode,v,vdot,nxi,xi,ucp,ucpx,ucpt,r,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
        Integer, Intent (Inout)              :: ires
        Integer, Intent (In)                 :: ncode, npde, nxi
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: r(ncode)
        Real (Kind=nag_wp), Intent (In)      :: ucp(npde,*), ucpt(npde,*),     &
                                                ucpx(npde,*), v(ncode),        &
                                                vdot(ncode), xi(nxi)
!       .. Executable Statements ..
        If (ires==-1) Then
          r(1) = 0.0_nag_wp
          r(2) = 0.0_nag_wp
        Else
          r(1) = v(1) - ucp(1,1) + ucp(2,1)
          r(2) = v(2) - ucp(1,2) - ucp(2,2)
        End If
        Return
      End Subroutine odedef
      Subroutine bndry2(npde,npts,t,x,u,ncode,v,vdot,ibnd,g,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t
        Integer, Intent (In)                 :: ibnd, ncode, npde, npts
        Integer, Intent (Inout)              :: ires
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: g(npde)
        Real (Kind=nag_wp), Intent (In)      :: u(npde,npts), v(ncode),        &
                                                vdot(ncode), x(npts)
!       .. Executable Statements ..
        If (ibnd==0) Then
          g(1) = u(1,1) - rl0
          g(2) = u(2,1)
          g(3) = u(3,1) - el0
        Else
          g(1) = u(1,npts) - rr0
          g(2) = u(2,npts)
          g(3) = u(3,npts) - er0
        End If
        Return
      End Subroutine bndry2
      Subroutine nmflx2(npde,t,x,ncode,v,uleft,uright,flux,ires)

!       .. Use Statements ..
```

```
        Use nag_library, Only: d03puf, d03pvf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: t, x
        Integer, Intent (Inout)              :: ires
        Integer, Intent (In)                 :: ncode, npde
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: flux(npde)
        Real (Kind=nag_wp), Intent (In)      :: uleft(npde), uright(npde),     &
                                                v(ncode)
!       .. Local Scalars ..
        Integer                              :: ifail
        Character (1)                        :: path, solver
!       .. Executable Statements ..
        ifail = 0
        solver = 'R'
        If (solver=='R') Then
!         ROE scheme ..
          Call d03puf(uleft,uright,gamma,flux,ifail)
        Else
!         OSHER scheme ..
          path = 'P'
          Call d03pvf(uleft,uright,gamma,path,flux,ifail)
        End If
        Return
      End Subroutine nmflx2
      Subroutine uvinit(npde,npts,x,u)

!       .. Scalar Arguments ..
        Integer, Intent (In)                 :: npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: u(npde,npts)
        Real (Kind=nag_wp), Intent (In)      :: x(npts)
!       .. Local Scalars ..
        Integer                              :: i
!       .. Executable Statements ..
        Do i = 1, npts
          If (x(i)<0.5_nag_wp) Then
            u(1,i) = rl0
            u(2,i) = 0.0_nag_wp
            u(3,i) = el0
          Else If (x(i)==0.5_nag_wp) Then
            u(1,i) = 0.5_nag_wp*(rl0+rr0)
            u(2,i) = 0.0_nag_wp
            u(3,i) = 0.5_nag_wp*(el0+er0)
          Else
            u(1,i) = rr0
            u(2,i) = 0.0_nag_wp
            u(3,i) = er0
          End If
        End Do
        Return
      End Subroutine uvinit
    End Module d03plfe_mod
    Program d03plfe

!     D03PLF Example Main Program

!     .. Use Statements ..
      Use d03plfe_mod, Only: nout
!     .. Implicit None Statement ..
      Implicit None
!     .. Executable Statements ..
      Write (nout,*) 'D03PLF Example Program Results'

      Call ex1

      Call ex2

    Contains
      Subroutine ex1
```

```
!       .. Use Statements ..
        Use nag_library, Only: d03plf, nag_wp
        Use d03plfe_mod, Only: bndry1, exact, itrace, ncode1, nin, nmflx1,     &
                               npde1, nxi1, odedef, pdedef
!       .. Local Scalars ..
        Real (Kind=nag_wp)                    :: errmax, lerr, lwgt, tout, ts
        Integer                               :: i, ifail, ind, itask, itol, j, &
                                                 lenode, lisave, lrsave, ncode, &
                                                 neqn, nfuncs, niters, njacs,   &
                                                 npde, npts, nsteps, nwkres, nxi
        Character (1)                         :: laopt, norm
!       .. Local Arrays ..
        Real (Kind=nag_wp)                    :: algopt(30), atol(1), rtol(1)
        Real (Kind=nag_wp), Allocatable       :: rsave(:), u(:), ue(:,:), x(:), &
                                                 xi(:)
        Integer, Allocatable                  :: isave(:)
!       .. Intrinsic Procedures ..
        Intrinsic                             :: abs, int, max, real
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*)
        Write (nout,*) 'Example 1'
        Write (nout,*)
!       Skip heading in data file
        Read (nin,*)
        Read (nin,*) npts
        npde = npde1
        ncode = ncode1
        nxi = nxi1
        neqn = npde*npts + ncode
        lisave = 25*neqn + 24
        nwkres = npde*(2*npts+6*nxi+3*npde+26) + nxi + ncode + 7*npts + 2
        lenode = 11*neqn + 50
        lrsave = 4*neqn + 11*neqn/2 + 1 + nwkres + lenode
        lisave = lisave*4
        lrsave = lrsave*4
        Allocate (rsave(lrsave),u(neqn),ue(npde,npts),x(npts),xi(nxi), &
          isave(lisave))

        Read (nin,*) itol
        Read (nin,*) norm
        Read (nin,*) atol(1), rtol(1)

!       Initialise mesh
        Do i = 1, npts
          x(i) = real(i-1,kind=nag_wp)/real(npts-1,kind=nag_wp)
        End Do
        xi(1) = 0.0_nag_wp
        xi(2) = 1.0_nag_wp

!       Set initial values ..
        ts = 0.0_nag_wp
        Call exact(ts,u,npde,x,npts)
        u(neqn-1) = u(1) - u(2)
        u(neqn) = u(neqn-2) + u(neqn-3)

        laopt = 'S'
        ind = 0
        itask = 1

        algopt(1:30) = 0.0_nag_wp
!       Theta integration
        algopt(1) = 1.0_nag_wp
!       Sparse matrix algebra parameters
        algopt(29) = 0.1_nag_wp
        algopt(30) = 1.1_nag_wp

        tout = 0.5_nag_wp

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
```

```
        ifail = 0
        Call d03plf(npde,ts,tout,pdedef,nmflx1,bndry1,u,npts,x,ncode,odedef, &
          nxi,xi,neqn,rtol,atol,itol,norm,laopt,algopt,rsave,lrsave,isave, &
          lisave,itask,itrace,ind,ifail)


        Write (nout,99992)
        Write (nout,99991) npts
        Write (nout,99990) rtol(1)
        Write (nout,99989) atol(1)

!       Calculate global error at t=tout : max (||u-ue||, over x)

!       Get exact solution at t=tout
        Call exact(tout,ue,npde,x,npts)
        errmax = -1.0_nag_wp
        Do i = 2, npts
          lerr = 0.0_nag_wp
          Do j = 1, npde
            lwgt = rtol(1)*abs(ue(j,i)) + atol(1)
            lerr = lerr + abs(u((i-1)*npde+j)-ue(j,i))/lwgt
          End Do
          lerr = lerr/real(npde,kind=nag_wp)
          errmax = max(errmax,lerr)
        End Do
        Write (nout,99999)
        Write (nout,99998) 100*int(errmax/100.0_nag_wp) + 100

!       Print integration statistics (reasonably rounded)
        nsteps = 50*((isave(1)+25)/50)
        nfuncs = 100*((isave(2)+50)/100)
        njacs = 20*((isave(3)+10)/20)
        niters = 100*((isave(5)+50)/100)
        Write (nout,99997)
        Write (nout,99996) nsteps
        Write (nout,99995) nfuncs
        Write (nout,99994) njacs
        Write (nout,99993) niters

        Return

99999   Format (/1X,'Integration Results:')
99998   Format (2X,'Global error is less than ',I3, &
          ' times the local error tolerance.')
99997   Format (/1X,'Integration Statistics:')
99996   Format (2X,'Number of time steps           (nearest  50) = ',I6)
99995   Format (2X,'Number of function evaluations (nearest 100) = ',I6)
99994   Format (2X,'Number of Jacobian evaluations (nearest  20) = ',I6)
99993   Format (2X,'Number of iterations           (nearest 100) = ',I6)
99992   Format (/1X,'Method Parameters:')
99991   Format (2X,'Number of mesh points used = ',I4)
99990   Format (2X,'Relative tolerance used    = ',E10.3)
99989   Format (2X,'Absolute tolerance used    = ',E10.3)
      End Subroutine ex1
      Subroutine ex2

!       .. Use Statements ..
        Use nag_library, Only: d03pek, d03plf, d03plp, nag_wp
        Use d03plfe_mod, Only: bndry2, el0, er0, gamma, itrace, ncode2, nin,   &
                               nmflx2, npde2, nxi2, rl0, rr0, uvinit
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: d, p, tout, ts, v
        Integer                              :: i, ifail, ind, it, itask,      &
                                                itol, lenode, lisave, lrsave,  &
                                                mlu, ncode, neqn, nfuncs,      &
                                                niters, njacs, npde, npts,     &
                                                nskip, nsteps, nwkres, nxi
        Character (1)                        :: laopt, norm
!       .. Local Arrays ..
        Real (Kind=nag_wp)                   :: algopt(30), atol(1), rtol(1),  &
                                                xi(1)
        Real (Kind=nag_wp), Allocatable      :: rsave(:), u(:,:), x(:)
        Integer, Allocatable                 :: isave(:)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: real
!       .. Executable Statements ..
        Write (nout,*)
        Write (nout,*)
        Write (nout,*) 'Example 2'
        Write (nout,*)
        Read (nin,*)
        Read (nin,*) npts, nskip

        npde = npde2
        ncode = ncode2
        nxi = nxi2
        nwkres = npde*(2*npts+3*npde+32) + 7*npts + 4
        mlu = 3*npde - 1
        neqn = npde*npts + ncode
        lenode = 9*neqn + 50
        lisave = neqn + 24
        lrsave = (3*mlu+1)*neqn + nwkres + lenode

        Allocate (rsave(lrsave),u(npde,npts),x(npts),isave(lisave))

!       Print problem parameters
        Write (nout,99997)
        Write (nout,99996) gamma
        Write (nout,99995) el0, er0
        Write (nout,99994) rl0, rr0

!       Read and print method parameters
        Read (nin,*) itol
        Read (nin,*) norm
        Read (nin,*) atol(1), rtol(1)

        Write (nout,99987)
        Write (nout,99986) npts
        Write (nout,99985) rtol(1)
        Write (nout,99984) atol(1)

!       Initialise mesh
        Do i = 1, npts
          x(i) = real(i-1,kind=nag_wp)/real(npts-1,kind=nag_wp)
        End Do

!       Initial values of variables
        Call uvinit(npde,npts,x,u)

        xi(1) = 0.0_nag_wp
        laopt = 'B'
        ind = 0
        itask = 1

        algopt(1:30) = 0.0_nag_wp
!       Theta integration
        algopt(1) = 2.0_nag_wp
        algopt(6) = 2.0_nag_wp
        algopt(7) = 2.0_nag_wp
!       Max. time step
        algopt(13) = 0.5E-2_nag_wp

        Write (nout,99999)
        Write (nout,99998)

        ts = 0.0_nag_wp
        tout = ts
        Do it = 1, 2
          tout = tout + 0.1_nag_wp

!         ifail: behaviour on error exit
!                =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
```

```
          ifail = 0
          Call d03plf(npde,ts,tout,d03plp,nmflx2,bndry2,u,npts,x,ncode,d03pek, &
            nxi,xi,neqn,rtol,atol,itol,norm,laopt,algopt,rsave,lrsave,isave, &
            lisave,itask,itrace,ind,ifail)

!         Calculate density, velocity and pressure ..
          Do i = 1, npts, nskip
            d = u(1,i)
            v = u(2,i)/d
            p = d*(gamma-1.0_nag_wp)*(u(3,i)/d-0.5_nag_wp*v**2)
            If (i==1) Then
              Write (nout,99993) ts, x(i), d, v, p
            Else
              Write (nout,99992) x(i), d, v, p
            End If
          End Do
          Write (nout,*)
        End Do

!       Print integration statistics (reasonably rounded)
        nsteps = 50*((isave(1)+25)/50)
        nfuncs = 50*((isave(2)+25)/50)
        njacs = isave(3)
        niters = isave(5)
        Write (nout,99991) nsteps
        Write (nout,99990) nfuncs
        Write (nout,99989) njacs
        Write (nout,99988) niters

        Return

99999   Format (/1X,'Solution')
99998   Format (4X,'t',6X,'x',5X,'density',1X,'velocity',1X,'pressure')
99997   Format (/' Problem Parameter and initial conditions:')
99996   Format ('  gamma          =',F6.3)
99995   Format ('      e(x<0.5,0) =',F6.3,'    e(x>0.5,0) =',F6.3)
99994   Format ('    rho(x>0.5,0) =',F6.3,'  rho(x>0.5,0) =',F6.3)
99993   Format (1X,F6.3,1X,F7.4,3(2X,F7.4))
99992   Format (8X,F7.3,3(2X,F7.4))
99991   Format (/' Number of time steps           (nearest 50) = ',I6)
99990   Format (' Number of function evaluations (nearest 50) = ',I6)
99989   Format (' Number of Jacobian evaluations (nearest  1) = ',I6)
99988   Format (' Number of iterations           (nearest  1) = ',I6)
99987   Format (/' Method Parameters:')
99986   Format ('  Number of mesh points used = ',I4)
99985   Format ('  Relative tolerance used    = ',E10.3)
99984   Format ('  Absolute tolerance used    = ',E10.3)
      End Subroutine ex2
    End Program d03plfe
```

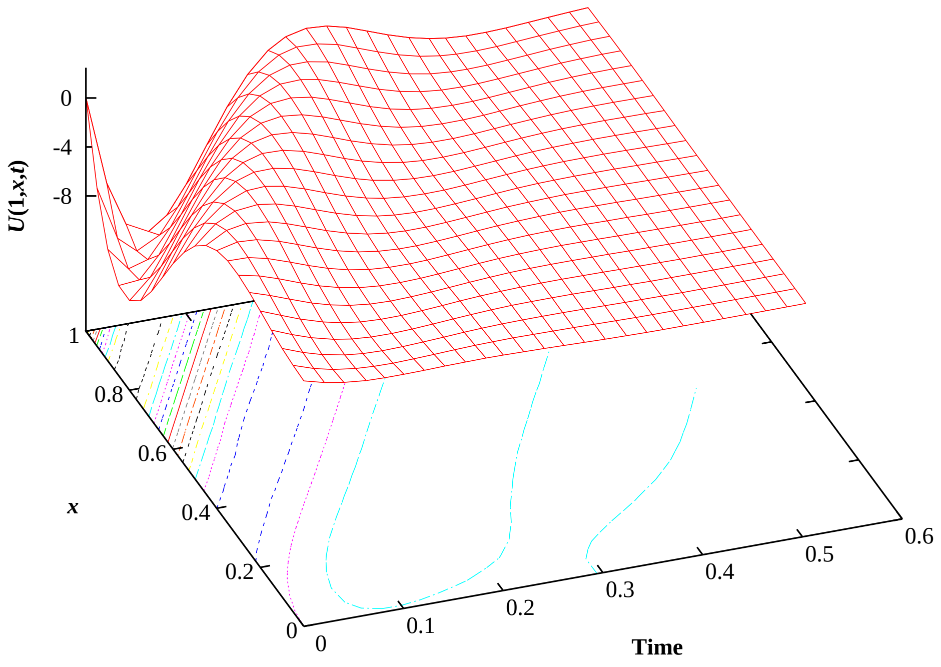
10.2 Program Data

```
D03PLF Example Program Data
  201                         : (ex1) npts
    1                         : itol
  '1'                         : norm
    0.1E-4  0.25E-3           : atol(1), rtol(1)

  141      14                 : (ex2) npts, nskip
    1                         : itol
  '2'                         : norm
    0.5E-2  0.5E-3            : atol(1), rtol(1)
```

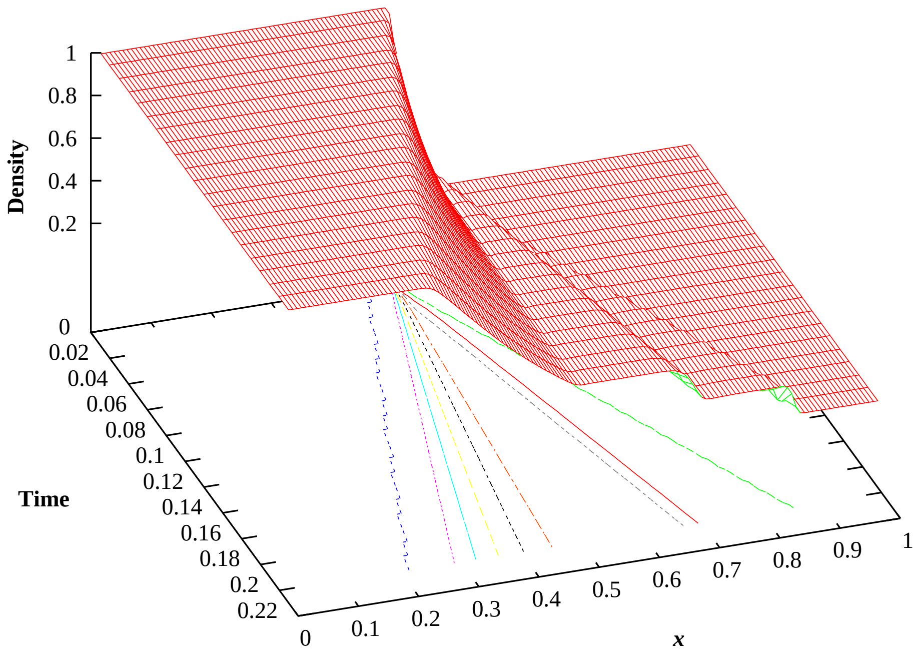
10.3 Program Results

```
D03PLF Example Program Results


Example 1


Method Parameters:
 Number of mesh points used =  201
 Relative tolerance used    =  0.250E-03
 Absolute tolerance used    =  0.100E-04

Integration Results:
 Global error is less than 100 times the local error tolerance.

Integration Statistics:
 Number of time steps           (nearest  50) =    150
 Number of function evaluations (nearest 100) =   1400
 Number of Jacobian evaluations (nearest  20) =     20
 Number of iterations           (nearest 100) =    400


Example 2


Problem Parameter and initial conditions:
 gamma          = 1.400
     e(x<0.5,0) = 2.500    e(x>0.5,0) = 0.250
   rho(x>0.5,0) = 1.000  rho(x>0.5,0) = 0.125

Method Parameters:
 Number of mesh points used =  141
 Relative tolerance used    =  0.500E-03
 Absolute tolerance used    =  0.500E-02

Solution
   t      x     density velocity pressure
 0.100  0.0000   1.0000   0.0000   1.0000
         0.100   1.0000  -0.0000   1.0000
         0.200   1.0000  -0.0000   1.0000
         0.300   1.0000  -0.0000   1.0000
         0.400   0.8668   0.1665   0.8188
         0.500   0.4299   0.9182   0.3071
         0.600   0.2969   0.9274   0.3028
         0.700   0.1250   0.0000   0.1000
         0.800   0.1250  -0.0000   0.1000
         0.900   0.1250  -0.0000   0.1000
         1.000   0.1250   0.0000   0.1000

 0.200  0.0000   1.0000   0.0000   1.0000
         0.100   1.0000  -0.0000   1.0000
         0.200   1.0000  -0.0000   1.0000
         0.300   0.8718   0.1601   0.8253
         0.400   0.6113   0.5543   0.5022
         0.500   0.4245   0.9314   0.3014
         0.600   0.4259   0.9277   0.3030
         0.700   0.2772   0.9272   0.3031
         0.800   0.2657   0.9276   0.3032
         0.900   0.1250  -0.0000   0.1000
         1.000   0.1250   0.0000   0.1000


Number of time steps           (nearest 50) =    150
Number of function evaluations (nearest 50) =    400
Number of Jacobian evaluations (nearest  1) =      1
Number of iterations           (nearest  1) =      2
```

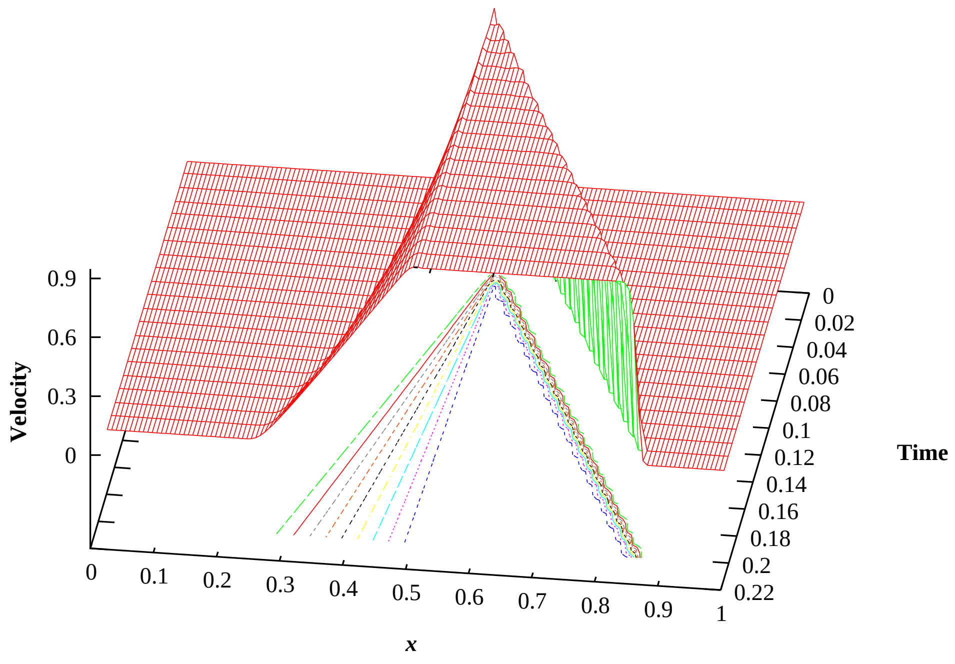
Example Program 1
First-order System with Coupled ODEs
Solution *U*(1,*x*,*t*)



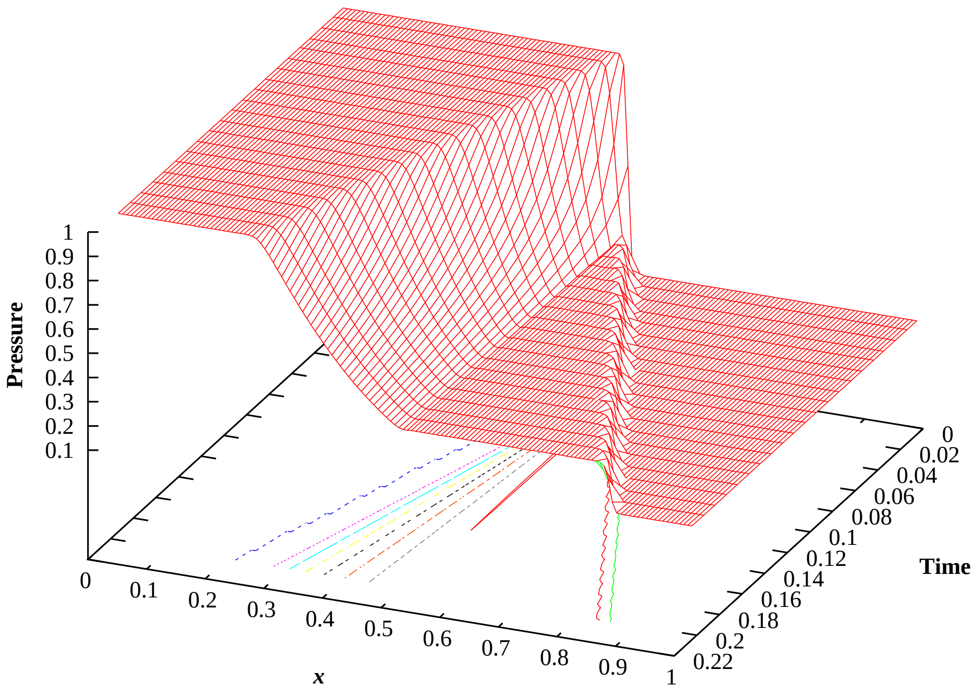
Example Program 2
Shock Tube Test Problem of Euler Equations in Gas Dynamics
DENSITY



Shock Tube Test Problem of Euler Equations in Gas Dynamics
VELOCITY



Shock Tube Test Problem of Euler Equations in Gas Dynamics
PRESSURE


Pressure
x
Time