NAG Toolbox

nag_pde_1d_parab_convdiff (d03pf)

1 Purpose

nag_pde_1d_parab_convdiff (d03pf) integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms. The system must be posed in conservative form. Convection terms are discretized using a sophisticated upwind scheme involving a user-supplied numerical flux function based on the solution of a Riemann problem at each mesh point. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs), and the resulting system is solved using a backward differentiation formula (BDF) method.

2 Syntax

```
[ts, u, rsave, isave, ind, ifail] = nag_pde_1d_parab_convdiff(ts, tout, pdedef,
numflx, bndary, u, x, acc, tsmax, rsave, isave, itask, itrace, ind, 'npde',
npde, 'npts', npts)
[ts, u, rsave, isave, ind, ifail] = d03pf(ts, tout, pdedef, numflx, bndary, u,
x, acc, tsmax, rsave, isave, itask, itrace, ind, 'npde', npde, 'npts', npts)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: *lrsave* and *lisave* were removed from the interface.

3 Description

nag_pde_1d_parab_convdiff (d03pf) integrates the system of convection-diffusion equations in conservative form:

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

or the hyperbolic convection-only system:

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

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

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

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

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, 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 $U(x,t)$ 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 nag_pde_1d_parab_fd (d03pc), nag_pde_1d_parab_dae_fd (d03ph) and nag_pde_1d_parab_remesh_fd (d03pp), but with the flux F_i replaced by a *numerical flux*, which is a representation of the flux taking into account the direction of the flow of information at that point (i.e., the direction of the characteristics). Simple

central differencing of the numerical flux then becomes a sophisticated upwind scheme in which the correct direction of upwinding is automatically achieved.

The numerical flux vector, $\hat{F}_i$ say, must be calculated by you in terms of the *left* and *right* values of the solution vector U (denoted by U_L and U_R respectively), at each mid-point of the mesh $x_{j-1/2} = \left(x_{j-1} + x_j\right)/2$, for $j = 2, 3, \ldots,$ **npts**. The left and right values are calculated by nag_pde_1d_parab_convdiff (d03pf) from two adjacent mesh points using a standard upwind technique combined with a Van Leer slope-limiter (see LeVeque (1990)). The physically correct value for $\hat{F}_i$ is derived from the solution of the Riemann problem given by

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

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

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

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

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

An example is given in Section 10.

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 **pdedef**. The numerical flux $\hat{F}_i$ must be supplied in a separate **numflx**. For problems in the form (2), the actual argument nag_pde_1d_parab_convdiff_sample_pdedef (d03pfp) may be used for **pdedef**. nag_pde_1d_parab_convdiff_sample_pdedef (d03pfp) is included in the NAG Toolbox and sets the matrix with entries $P_{i,j}$ to the identity matrix, and the functions C_i, D_i and S_i to zero.

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

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

A common way of providing numerical boundary conditions is to extrapolate the characteristic variables from the inside of the domain. Note that only linear extrapolation is allowed in this function (for greater flexibility the function nag_pde_1d_parab_convdiff_dae (d03pl) should be used). For problems in which the solution is known to be uniform (in space) towards a boundary during the period

of integration then extrapolation is unnecessary; the numerical boundary condition can be supplied as the known solution at the boundary. Examples can be found in Section 10.

The boundary conditions must be specified in **bndary** in the form

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

at the left-hand boundary, and

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

at the right-hand boundary.

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

The problem is subject to the following restrictions:

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

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

(iii) $t_0 < t_{\text{out}}$, 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 $x_1, x_2, \ldots, x_{\mathbf{npts}}$;

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

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

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

4 References

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

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

LeVeque R J (1990) *Numerical Methods for Conservation Laws* Birkhìuser Verlag

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

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

5 Parameters

5.1 Compulsory Input Parameters

1: **ts** – REAL (KIND=nag_wp)

The initial value of the independent variable t.

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

2: **tout** – REAL (KIND=nag_wp)

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

3: **pdedef** – SUBROUTINE, supplied by the NAG Library or the user.

pdedef must evaluate the functions $P_{i,j}$, C_i, D_i and S_i which partially define the system of PDEs. $P_{i,j}$, C_i and S_i may depend on x, t and U; D_i may depend on x, t, U and U_x. **pdedef** is called approximately midway between each pair of mesh points in turn by nag_pde_1d_parab_convdiff (d03pf). The actual argument nag_pde_1d_parab_convdiff_sample_pdedef (d03pfp) may be used for **pdedef** for problems in the form (2). (nag_pde_1d_parab_convdiff_sample_pdedef (d03pfp) is included in the NAG Toolbox.)

```
[p, c, d, s, ires] = pdedef(npde, t, x, u, ux, ires)
```

Input Parameters

1: **npde** – INTEGER

The number of PDEs in the system.

2: **t** – REAL (KIND=nag_wp)

The current value of the independent variable t.

3: **x** – REAL (KIND=nag_wp)

The current value of the space variable x.

4: **u**(**npde**) – REAL (KIND=nag_wp) array

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

5: **ux**(**npde**) – REAL (KIND=nag_wp) array

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

6: **ires** – INTEGER

Set to -1 or 1.

Output Parameters

1: **p**(**npde**, **npde**) – REAL (KIND=nag_wp) array

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

2: **c**(**npde**) – REAL (KIND=nag_wp) array

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

3: **d**(**npde**) – REAL (KIND=nag_wp) array

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

4: **s**(**npde**) – REAL (KIND=nag_wp) array

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

5: **ires** – INTEGER

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 function 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 nag_pde_1d_parab_convdiff (d03pf) returns to the calling function with the error indicator set to **ifail** $= 4$.

4: **numflx** – SUBROUTINE, supplied by the user.

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 nag_pde_1d_parab_convdiff (d03pf).

```
[flux, ires] = numflx(npde, t, x, uleft, uright, ires)
```

Input Parameters

1: **npde** – INTEGER

The number of PDEs in the system.

2: **t** – REAL (KIND=nag_wp)

The current value of the independent variable t.

3: **x** – REAL (KIND=nag_wp)

The current value of the space variable x.

4: **uleft**(**npde**) – REAL (KIND=nag_wp) array

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

5: **uright**(**npde**) – REAL (KIND=nag_wp) array

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

6: **ires** – INTEGER

Set to -1 or 1.

Output Parameters

1: **flux**(**npde**) – REAL (KIND=nag_wp) array

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

2: **ires** – INTEGER

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 function 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 nag_pde_1d_parab_convdiff (d03pf) returns to the calling function with the error indicator set to **ifail** $= 4$.

5: **bndary** – SUBROUTINE, supplied by the user.

bndary must evaluate the functions G_i^L and G_i^R which describe the physical and numerical boundary conditions, as given by (6) and (7).

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

Input Parameters

1: **npde** – INTEGER

The number of PDEs in the system.

2: **npts** – INTEGER

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

3: **t** – REAL (KIND=nag_wp)

The current value of the independent variable t.

4: **x**(**npts**) – REAL (KIND=nag_wp) array

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

5: **u**(**npde**, **3**) – REAL (KIND=nag_wp) array

Contains the value of solution components in the boundary region.

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

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

6: **ibnd** – INTEGER

Specifies which boundary conditions are to be evaluated.

ibnd $= 0$
bndary must evaluate the left-hand boundary condition at $x = a$.

ibnd $\neq 0$
bndary must evaluate the right-hand boundary condition at $x = b$.

7: **ires** – INTEGER

Set to -1 or 1.

Output Parameters

1: **g**(**npde**) – REAL (KIND=nag_wp) array

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

2: **ires** – INTEGER

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 function 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 nag_pde_1d_parab_convdiff (d03pf) returns to the calling function with the error indicator set to **ifail** $= 4$.

6: **u**(**npde**, **npts**) – REAL (KIND=nag_wp) array

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

7: **x**(**npts**) – REAL (KIND=nag_wp) array

The mesh points in the space direction. **x**(1) must specify the left-hand boundary, a, and **x**(**npts**) must specify the right-hand boundary, b.

Constraint: **x**$(1) <$ **x**$(2) < \cdots <$ **x**(**npts**).

8: **acc**(**2**) – REAL (KIND=nag_wp) array

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

If E(i, j) is the estimated error for U_i at the jth mesh point, the error test is

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

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

9: **tsmax** – REAL (KIND=nag_wp)

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

Constraint: **tsmax** ≥ 0.0.

10: **rsave**$(lrsave)$ – REAL (KIND=nag_wp) array

$lrsave$, the dimension of the array, must satisfy the constraint $lrsave \geq (11 + 9 \times$ **npde**$) \times$ **npde** $\times$ **npts** $+ (32 + 3 \times$ **npde**$) \times$ **npde** $+ 7 \times$ **npts** $+ 54$.

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

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

11: **isave**$(lisave)$ – INTEGER array

$lisave$, the dimension of the array, must satisfy the constraint $lisave \geq$ **npde** $\times$ **npts** $+ 24$.

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

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

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

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

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

isave(4)
Contains the order of the last backward differentiation formula method used.

isave(5)
Contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the LU decomposition of the Jacobian matrix.

12: **itask** – INTEGER

The task to be performed by the ODE integrator.

itask $= 1$
Normal computation of output values **u** at $t =$ **tout** (by overshooting and interpolating).

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

itask $= 3$
Stop at first internal integration point at or beyond $t =$ **tout**.

Constraint: **itask** $= 1$, 2 or 3.

13: **itrace** – INTEGER

The level of trace information required from nag_pde_1d_parab_convdiff (d03pf) and the underlying ODE solver. **itrace** may take the value -1, 0, 1, 2 or 3.

itrace $= -1$
No output is generated.

itrace $= 0$
Only warning messages from the PDE solver are printed on the current error message unit (see nag_file_set_unit_error (x04aa)).

itrace > 0
Output from the underlying ODE solver is printed on the current advisory message unit (see nag_file_set_unit_advisory (x04ab)). This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

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

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

14: **ind** – INTEGER

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 function. In this case, only the arguments **tout** and **ifail** should be reset between calls to nag_pde_1d_parab_convdiff (d03pf).

Constraint: **ind** $= 0$ or 1.

5.2 Optional Input Parameters

1: **npde** – INTEGER

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

The number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **npts** – INTEGER

Default: the dimension of the array **x** and the second dimension of the array **u**. (An error is raised if these dimensions are not equal.)

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

Constraint: **npts** ≥ 3.

5.3 Output Parameters

1: **ts** – REAL (KIND=nag_wp)

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

2: **u**(**npde**, **npts**) – REAL (KIND=nag_wp) array

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

3: **rsave**$(lrsave)$ – REAL (KIND=nag_wp) array

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

4: **isave**$(lisave)$ – INTEGER array

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

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

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

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

isave(4)
Contains the order of the last backward differentiation formula method used.

isave(5)
Contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the LU decomposition of the Jacobian matrix.

5: **ind** – INTEGER

ind $= 1$.

6: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, **ts** $\geq$ **tout**,
or **tout** $-$ **ts** is too small,
or **itask** $= 1$, 2 or 3,
or **npts** < 3,
or **npde** < 1,
or **ind** $\neq 0$ or 1,
or incorrect user-defined mesh, i.e., $\mathbf{x}(i) \geq \mathbf{x}(i+1)$ for some $i = 1, 2, \ldots,$ **npts** $- 1$,
or $lrsave$ or $lisave$ are too small,
or **ind** $= 1$ on initial entry to nag_pde_1d_parab_convdiff (d03pf),
or **acc**(1) or **acc**$(2) < 0.0$,
or **acc**(1) or **acc**(2) are both zero,
or **tsmax** < 0.0.

ifail $= 2$ (*warning*)

The underlying ODE solver cannot make any further progress, with the values of **acc**, across the integration range from the current point $t =$ **ts**. The components of **u** contain the computed values at the current point $t =$ **ts**.

ifail $= 3$ (*warning*)

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 **pdedef**, **numflx** or **bndary** when the residual in the underlying ODE solver was being evaluated. Incorrect specification of boundary conditions may also result in this error.

ifail $= 5$

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

ifail $= 6$ (*warning*)

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

ifail $= 7$

The values of **acc**(1) and **acc**(2) are so small that the function is unable to start the integration in time.

ifail $= 8$

In either, **pdedef**, **numflx** or **bndary**, **ires** was set to an invalid value.

ifail $= 9$ (nag_ode_ivp_stiff_imp_revcom (d02nn))

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

ifail $= 10$ (*warning*)

The required task has been completed, but it is estimated that a small change in the values of **acc** is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when **itask** $\neq 2$.)

ifail $= 11$

An error occurred during Jacobian formulation of the ODE system (a more detailed error description may be directed to the current advisory message unit when **itrace** ≥ 1).

ifail $= 12$

Not applicable.

ifail $= 13$

Not applicable.

ifail $= 14$

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

ifail $= -99$

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

ifail $= -399$

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

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

nag_pde_1d_parab_convdiff (d03pf) controls the accuracy of the integration in the time direction but not the accuracy of the approximation in space. The spatial accuracy depends on both the number of mesh points and on their distribution in space. In the time integration only the local error over a single step is controlled and so the accuracy over a number of steps cannot be guaranteed. You should therefore test the effect of varying the components of the accuracy argument, **acc**.

8 Further Comments

nag_pde_1d_parab_convdiff (d03pf) is designed to solve systems of PDEs in conservative form, with optional source terms which are independent of space derivatives, and optional second-order diffusion terms. The use of the function to solve systems which are not naturally in this form is discouraged, and you are advised to use one of the central-difference schemes for such problems.

You should be aware of the stability limitations for hyperbolic PDEs. For most problems with small error tolerances the ODE integrator does not attempt unstable time steps, but in some cases a maximum time step should be imposed using **tsmax**. It is worth experimenting with this argument, particularly if

the integration appears to progress unrealistically fast (with large time steps). Setting the maximum time step to the minimum mesh size is a safe measure, although in some cases this may be too restrictive.

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

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

9 Example

For this function two examples are presented. There is a single example program for nag_pde_1d_parab_convdiff (d03pf), with a main program and the code to solve the two example problems given in Example 1 (EX1) and Example 2 (EX2).

Example 1 (EX1)

This example is a simple first-order system which illustrates the calculation of the numerical flux using Roe's approximate Riemann solver, and the specification of numerical boundary conditions using extrapolated characteristic variables. The PDEs are

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

for $x \in [0,1]$ and $t \geq 0$. The PDEs have an exact solution given by

$$U_1(x,t) = \tfrac{1}{2}\{\exp(x+t)+\exp(x-3t)\}+\tfrac{1}{4}\left\{\sin\left(2\pi(x-3t)^2\right)-\sin\left(2\pi(x+t)^2\right)\right\}+2t^2-2xt,$$

$$U_2(x,t) = \exp(x-3t)-\exp(x+t)+\tfrac{1}{2}\left\{\sin\left(2\pi(x-3t)^2\right)+\sin\left(2\pi(x-3t)^2\right)\right\}+x^2+5t^2-2xt.$$

The initial conditions are given by the exact solution. The characteristic variables are $2U_1+U_2$ and $2U_1-U_2$ corresponding to the characteristics given by $dx/dt=3$ and $dx/dt=-1$ respectively. Hence a physical boundary condition is required for $2U_1+U_2$ at the left-hand boundary, and for $2U_1-U_2$ at the right-hand boundary (corresponding to the incoming characteristics); and a numerical boundary condition is required for $2U_1-U_2$ at the left-hand boundary, and for $2U_1+U_2$ at the right-hand boundary (outgoing characteristics). The physical boundary conditions are obtained from the exact solution, and the numerical boundary conditions are calculated by linear extrapolation of the appropriate characteristic variable. The numerical flux is calculated using Roe's approximate Riemann solver: Using the notation in Section 3, the flux vector F and the Jacobian matrix A are

$$F=\begin{bmatrix} U_1+U_2 \\ 4U_1+U_2 \end{bmatrix} \quad \text{and} \quad A=\begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix},$$

and the eigenvalues of A are 3 and -1 with right eigenvectors $\begin{bmatrix} 1 & 2 \end{bmatrix}^{\mathrm{T}}$ and $\begin{bmatrix} -1 & 2 \end{bmatrix}^{\mathrm{T}}$ respectively. Using equation (4) the α_k are given by

$$\begin{bmatrix} U_{1R}-U_{1L} \\ U_{2R}-U_{2L} \end{bmatrix} = \alpha_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \alpha_2 \begin{bmatrix} -1 \\ 2 \end{bmatrix},$$

that is

$$\alpha_1 = \tfrac{1}{4}(2U_{1R}-2U_{1L}+U_{2R}-U_{2L}) \quad \text{and} \quad \alpha_2 = \tfrac{1}{4}(-2U_{1R}+2U_{1L}+U_{2R}-U_{2L}).$$

F_L is given by

$$F_L=\begin{bmatrix} U_{1L}+U_{2L} \\ 4U_{1L}+U_{2L} \end{bmatrix},$$

and similarly for F_R. From equation (4), the numerical flux vector is

$$\hat{F} = \tfrac{1}{2}\begin{bmatrix} U_{1L} + U_{2L} + U_{1R} + U_{2R} \\ 4U_{1L} + U_{2L} + 4U_{1R} + U_{2R} \end{bmatrix} - \tfrac{1}{2}\alpha_1|3|\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \tfrac{1}{2}\alpha_2|-1|\begin{bmatrix} -1 \\ 2 \end{bmatrix},$$

that is

$$\hat{F} = \tfrac{1}{2}\begin{bmatrix} 3U_{1L} - U_{1R} + \frac{3}{2}U_{2L} + \frac{1}{2}U_{2R} \\ 6U_{1L} + 2U_{1R} + 3U_{2L} - U_{2R} \end{bmatrix}.$$

Example 2 (EX2)

This example is an advection-diffusion equation in which the flux term depends explicitly on x:

$$\frac{\partial U}{\partial t} + x\frac{\partial U}{\partial x} = \epsilon\frac{\partial^2 U}{\partial x^2},$$

for $x \in [-1, 1]$ and $0 \leq t \leq 10$. The argument ϵ is taken to be 0.01. The two physical boundary conditions are $U(-1, t) = 3.0$ and $U(1, t) = 5.0$ and the initial condition is $U(x, 0) = x + 4$. The integration is run to steady state at which the solution is known to be $U = 4$ across the domain with a narrow boundary layer at both boundaries. In order to write the PDE in conservative form, a source term must be introduced, i.e.,

$$\frac{\partial U}{\partial t} + \frac{\partial (xU)}{\partial x} = \epsilon\frac{\partial^2 U}{\partial x^2} + U.$$

As in Example 1, the numerical flux is calculated using the Roe approximate Riemann solver. The Riemann problem to solve locally is

$$\frac{\partial U}{\partial t} + \frac{\partial (xU)}{\partial x} = 0.$$

The x in the flux term is assumed to be constant at a local level, and so using the notation in Section 3, $F = xU$ and $A = x$. The eigenvalue is x and the eigenvector (a scalar in this case) is 1. The numerical flux is therefore

$$\hat{F} = \begin{cases} xU_L & \text{if } x \geq 0, \\ xU_R & \text{if } x < 0. \end{cases}$$

9.1 Program Text

```
     function d03pf_example

fprintf('d03pf example results\n\n');

fprintf('Example 1\n');
ex1;
fprintf('\n\nExample 2\n');
ex2;

function ex1
  ts   = 0;
  tout = 0.1;
  npde = 2;
  npts = 101;
  x    = [0:0.01:1];
  u    = exact(ts,npde,x,npts);
  acc  = [1e-04;  1e-05];
  tsmax  = 0;
  rsave  = zeros(6695, 1);
  isave  = zeros(226, 1, nag_int_name);
  itask  = nag_int(1);
  itrace = nag_int(0);
  ind    = nag_int(0);

  [ts, u, rsave, isave, ind, ifail] = ...
  d03pf( ...
         ts, tout, @pdedef, @numflx1, @bndary1, u, x, acc, ...
```

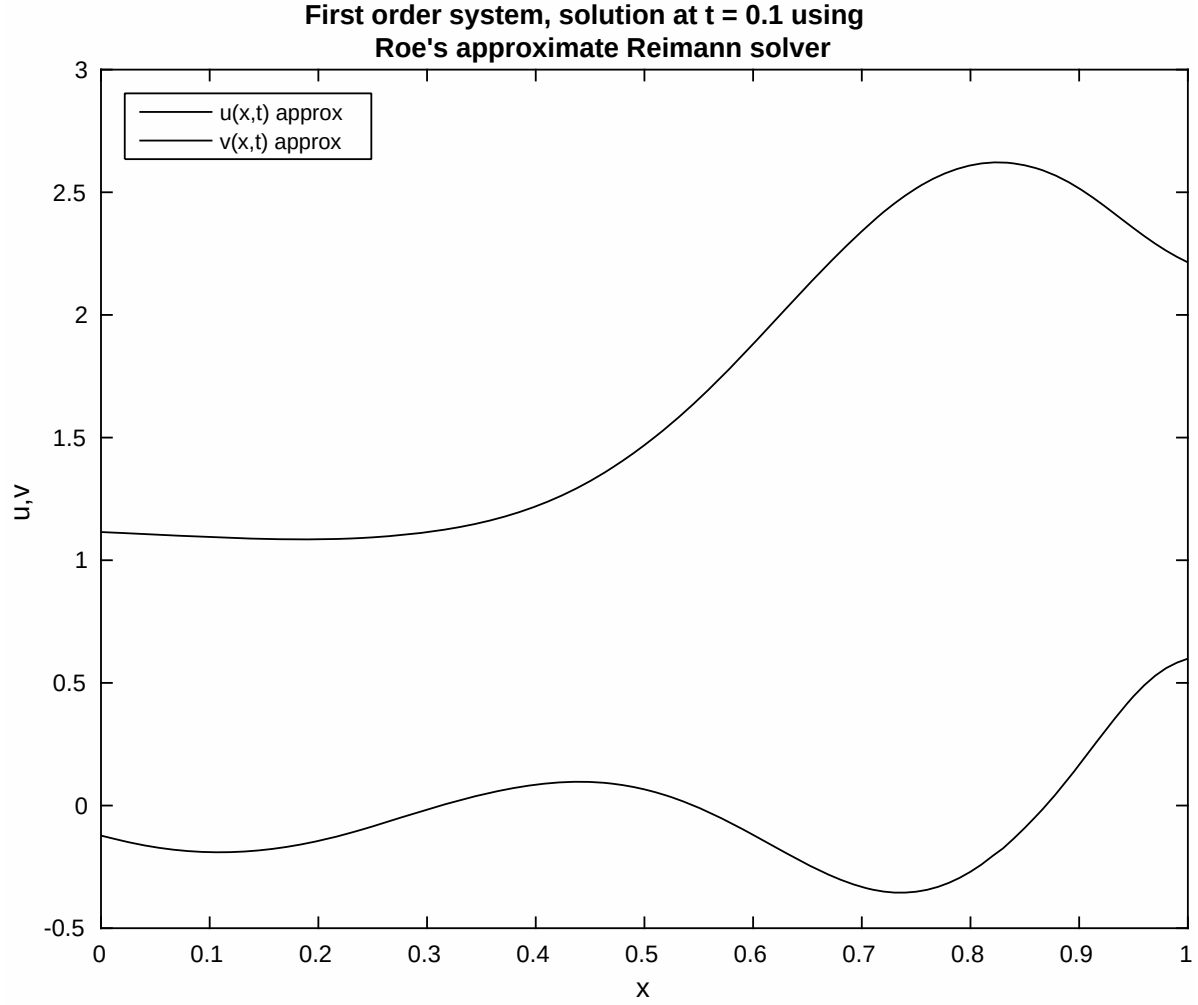
```
        tsmax, rsave, isave, itask, itrace, ind);

  uerr  = exact(ts,npde,x,npts) - u;
  l2err = norm(uerr);

  fprintf(' At t = %7.4f the L2-error in computed solution = %7.4f\n', ...
          ts, l2err);

  nsteps = 5*((isave(1)+2)/5);
  nfuncs = 50*((isave(2)+25)/50);
  njacs = 10*((isave(3)+5)/10);
  niters = 50*((isave(5)+25)/50);
  fprintf('\n Number of time steps           (nearest 5)  = %6d\n',nsteps);
  fprintf(' Number of function evaluations (nearest 50) = %6d\n',nfuncs);
  fprintf(' Number of Jacobian evaluations (neaerst 10) = %6d\n',njacs);
  fprintf(' Number of iterations           (nearest 50) = %6d\n',niters);

  fig1 = figure;
  plot(x,u(1,:),x,u(2,:));
  title({['First order system, solution at t = 0.1 using'],...
         ['Roe''s approximate Reimann solver']});
  xlabel('x');
  ylabel('u,v');
  legend('u(x,t) approx','v(x,t) approx','Location','NorthWest');
  % print(fig1,'-dpng','-r75','d03pf_fig1.png');
  % print(fig1,'-deps','-r75','d03pf_fig1.eps');

function ex2
  ts   = 0;
  tout = 1;
  npde = 1;
  npts = 151;
  dx   = 2/(npts-1);
  x    = [-1:dx:1];
  u(1,:) = x + 4;
  acc  = [1e-05;  1e-05];
  tsmax  = 0.02;
  rsave  = zeros(6695, 1);
  isave  = zeros(1000, 1, nag_int_name);
  itask  = nag_int(1);
  itrace = nag_int(0);
  ind    = nag_int(0);

  [ts, u, rsave, isave, ind, ifail] = ...
  d03pf( ...
         ts, tout, @pdedef, @numflx2, @bndary2, u, x, acc, ...
         tsmax, rsave, isave, itask, itrace, ind);

  nsteps = 5*((isave(1)+2)/5);
  nfuncs = 50*((isave(2)+25)/50);
  njacs = 10*((isave(3)+5)/10);
  niters = 50*((isave(5)+25)/50);
  fprintf('\n Number of time steps           (nearest 5)  = %6d\n',nsteps);
  fprintf(' Number of function evaluations (nearest 50) = %6d\n',nfuncs);
  fprintf(' Number of Jacobian evaluations (neaerst 10) = %6d\n',njacs);
  fprintf(' Number of iterations           (nearest 50) = %6d\n',niters);

  fig2 = figure;
  plot(x,u(1,:));
  title({['Burger''s equation, solution at t = 0.1 using'],...
         ['Roe''s approximate Reimann solver']});
  xlabel('x');
  ylabel('u,v');
  % print(fig2,'-dpng','-r75','d03pf_fig2.png');
  % print(fig2,'-deps','-r75','d03pf_fig2.eps');

function [p, c, d, s, ires] = pdedef(npde, t, x, u, ux, ires)
    %  simple hyperbolic system.
    p = eye(npde);
    c = zeros(npde, 1); d = c; s = c;
    c(1) = 0.01;
```

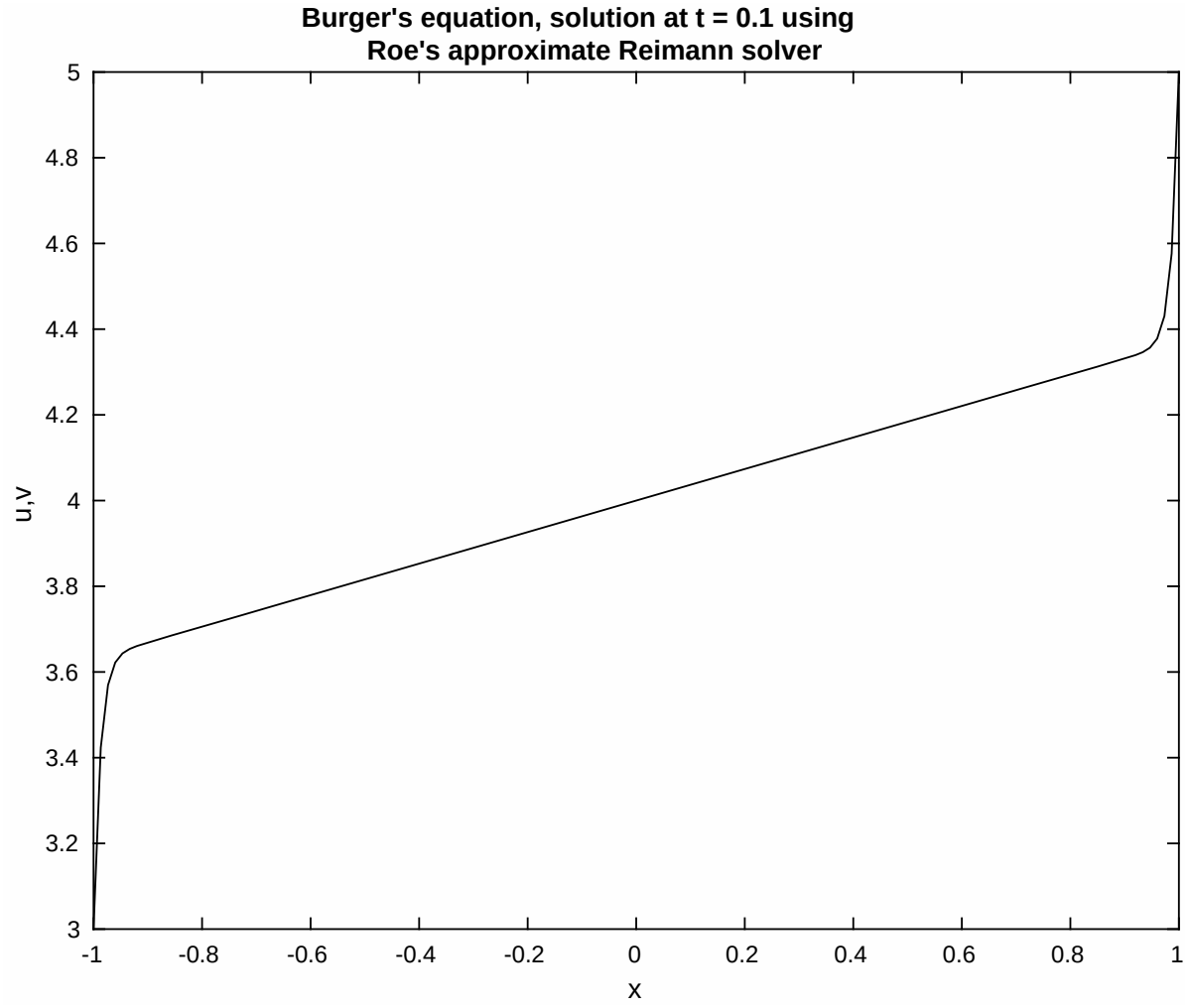
```
    d(1) = ux(1);
    s(1) = u(1);

function [flux,ires] = numflx1(npde, t, x, uleft, uright, ires)
  flux = zeros(npde,1);
  flux(1) = (-2*uright(1)+6*uleft(1)+uright(2)+3*uleft(2))/4;
  flux(2) = ( 2*uright(1)+6*uleft(1)-uright(2)+3*uleft(2))/2;

function [flux,ires] = numflx2(npde, t, x, uleft, uright, ires)
  if (x>=0)
    flux(1) = x*uleft(1);
  else
    flux(1) = x*uright(1);
  end

function [g, ires] = bndary1(npde, npts, t, x, u, ibnd, ires)
  np1 = npts-1;
  np2 = npts-2;
  g   = zeros(npde,1);
  if (ibnd == 0)
    ue   = exact(t,npde,x(1),1);
    c    = (x(2)-x(1))/(x(3)-x(2));
    exu1 = (1+c)*u(1,2) - c*u(1,3);
    exu2 = (1+c)*u(2,2) - c*u(2,3);
    g(1) = 2*u(1,1) + u(2,1) - 2*ue(1,1) - ue(2,1);
    g(2) = 2*u(1,1) - u(2,1) - 2*exu1 + exu2;
  else
    ue   = exact(t,npde,x(npts),1);
    c    = (x(npts)-x(np1))/(x(np1)-x(np2));
    exu1 = (1+c)*u(1,2) - c*u(1,3);
    exu2 = (1+c)*u(2,2) - c*u(2,3);
    g(1) = 2*u(1,1) - u(2,1) - 2*ue(1,1) + ue(2,1);
    g(2) = 2*u(1,1) + u(2,1) - 2*exu1 - exu2;
  end

function [g, ires] = bndary2(npde, npts, t, x, u, ibnd, ires)
  np1 = npts-1;
  np2 = npts-2;
  g   = zeros(npde,1);
  if (ibnd == 0)
    g(1) = u(1,1) - 3;
  else
    g(1) = u(1,1) - 5;
  end

% exact solution (for comparison and b.c. purposes)
function u = exact(t,npde,x,npts)
  u = zeros(npde,npts);
  for i = 1:double(npts)
    x1 = x(i) + t;
    x2 = x(i) - 3*t;
    s1 = sin(2*pi*x1^2)/2;
    s2 = sin(2*pi*x2^2)/2;

    u(1,i) = (exp(x1)+exp(x2))/2 + (s2-s1)/2 + 2*t^2 - 2*x(i)*t;
    u(2,i) = exp(x2)-exp(x1) + (s2+s1) + x(i)^2 + 5*t^2 - 2*x(i)*t;
  end
```

9.2 Program Results

```
     d03pf example results

Example 1
 At t =  0.1000 the L2-error in computed solution =  1.5889

 Number of time steps           (nearest 5)  =     35
 Number of function evaluations (nearest 50) =    200
 Number of Jacobian evaluations (neaerst 10) =     10
 Number of iterations           (nearest 50) =    100
```

```
Example 2

 Number of time steps           (nearest 5)  =     55
 Number of function evaluations (nearest 50) =    200
 Number of Jacobian evaluations (neaerst 10) =     10
 Number of iterations           (nearest 50) =    150
```




Burger's equation, solution at t = 0.1 using
Roe's approximate Reimann solver
u,v
x
5
4.8
4.6
4.4
4.2
4
3.8
3.6
3.4
3.2
3
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1