NAG C Library Function Document

nag_pde_parab_1d_fd_ode (d03phc)

1 Purpose

nag_pde_parab_1d_fd_ode (d03phc) integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs). The spatial discretisation is performed using finite differences, and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a backward differentiation formula method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
void nag_pde_parab_1d_fd_ode (Integer npde, Integer m, double *ts, double tout,
     void (*pdedef)(Integer npde, double t, double x, const double u[],
          const double ux[], Integer ncode, const double v[], const double vdot[],
          double p[], double q[], double r[], Integer *ires, Nag_Comm *comm),
     void (*bndary)(Integer npde, double t, const double u[], const double ux[],
          Integer ncode, const double v[], const double vdot[], Integer ibnd,
          double beta[], double gamma[], Integer *ires, Nag_Comm *comm),
     double u[], Integer npts, const double x[], Integer ncode,
     void (*odedef)(Integer npde, double t, Integer ncode, const double v[],
          const double vdot[], Integer nxi, const double xi[], const double ucp[],
          const double ucpx[], const double rcp[], const double ucpt[],
          const double ucptx[], double f[], Integer *ires, Nag_Comm *comm),
     Integer nxi, const double xi[], Integer neqn, const double rtol[],
     const double atol[], Integer itol, Nag_NormType norm, Nag_LinAlgOption laopt,
     const double algopt[], double rsave[], Integer lrsave, Integer isave[],
     Integer lisave, Integer itask, Integer itrace, const char *outfile, Integer *ind,
     Nag_Comm *comm, Nag_D03_Save *saved, NagError *fail)
```

3 Description

nag_pde_parab_1d_fd_ode (d03phc) integrates the system of parabolic-elliptic equations and coupled ODEs

$$\sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + Q_i = x^{-m}\frac{\partial}{\partial x}(x^m R_i), \quad i = 1,2,\ldots,\mathbf{npde}, \quad a \le x \le b, \quad t \ge t_0, \tag{1}$$

$$F_i(t, V, \dot{V}, \xi, U^*, U_x^*, R^*, U_t^*, U_{xt}^*) = 0, \quad i = 1,2,\ldots,\mathbf{ncode}, \tag{2}$$

where (1) defines the PDE part and (2) generalizes the coupled ODE part of the problem.

In (1), $P_{i,j}$ and R_i depend on x, t, U, U_x and V; Q_i depends on x, t, U, U_x, V and **linearly** on $\dot{V}$. The vector U is the set of PDE solution values

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

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

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

and $\dot{V}$ denotes its derivative with respect to time.

In (2), ξ represents a vector of n_ξ spatial coupling points at which the ODEs are coupled to the PDEs. These points may or may not be equal to some of the PDE spatial mesh points. U^*, U_x^*, R^*, U_t^* and U_{xt}^*

are the functions U, U_x, R, U_t and U_{xt} evaluated at these coupling points. Each F_i may only depend linearly on time derivatives. Hence the equation (2) may be written more precisely as

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{3}$$

where $F = [F_1, \ldots, F_{\mathbf{ncode}}]^T$, G is a vector of length **ncode**, A is an **ncode** by **ncode** matrix, B is an **ncode** by ($n_\xi \times$ **npde**) matrix and the entries in G, A and B may depend on t, ξ, U^*, U_x^* and V. In practice the user only needs to supply a vector of information to define the ODEs and not the matrices A and B. (See Section 5 for the specification of the user-supplied procedure **odedef**.)

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$. The co-ordinate system in space is defined by the values of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates.

The PDE system which is defined by the functions $P_{i,j}$, Q_i and R_i must be specified in a function **pdedef** supplied by the user.

The initial values of the functions $U(x, t)$ and $V(t)$ must be given at $t = t_0$.

The functions R_i which may be thought of as fluxes, are also used in the definition of the boundary conditions. The boundary conditions must have the form

$$\beta_i(x,t)R_i(x,t,U,U_x,V) = \gamma_i(x,t,U,U_x,V,\dot{V}), \quad i = 1, 2, \ldots, \mathbf{npde}, \tag{4}$$

where $x = a$ or $x = b$.

The boundary conditions must be specified in a function **bndary** provided by the user. The function γ_i may depend **linearly** on $\dot{V}$.

The problem is subject to the following restrictions:

(i) In (1), $\dot{V}_j(t)$, for $j = 1, 2, \ldots,$ **ncode**, may only appear **linearly** in the functions Q_i, for $i = 1, 2, \ldots,$ **npde**, with a similar restriction for γ;

(ii) $P_{i,j}$ and the flux R_i must not depend on any time derivatives;

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

(iv) the evaluation of the terms $P_{i,j}$, Q_i and R_i is done approximately at the mid-points of the mesh **x**$[i-1]$, for $i = 1, 2, \ldots,$ **npts**, by calling the function **pdedef** for each mid-point 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 non-zero so that there is a time derivative present in the PDE problem;

(vi) if $m > 0$ and $x_1 = 0.0$, which is the left boundary point, then it must be ensured that the PDE solution is bounded at this point. This can be done by either specifying the solution at $x = 0.0$ or by specifying a zero flux there, that is $\beta_i = 1.0$ and $\gamma_i = 0.0$. See also Section 8 below.

The algebraic-differential equation system which is defined by the functions F_i must be specified in a function **odedef** supplied by the user. The user must also specify the coupling points ξ in the array **xi**.

The parabolic equations are approximated by a system of ODEs in time for the values of U_i at mesh points. For simple problems in Cartesian co-ordinates, this system is obtained by replacing the space derivatives by the usual central, three-point finite-difference formula. However, for polar and spherical problems, or problems with nonlinear coefficients, the space derivatives are replaced by a modified three-point formula which maintains second order accuracy. In total there are **npde** $\times$ **npts** $+$ **ncode** ODEs in the time direction. This system is then integrated forwards in time using a backward differentiation formula (BDF) or a Theta method.

4 References

Berzins M (1990) Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (ed J C Mason and M G Cox) 59–72 Chapman and Hall

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

Berzins M and Furzeland R M (1992) An adaptive theta method for the solution of stiff and nonstiff differential equations *Appl. Numer. Math.* **9** 1–19

Skeel R D and Berzins M (1990) A method for the spatial discretization of parabolic equations in one space variable *SIAM J. Sci. Statist. Comput.* **11 (1)** 1–32

5 Parameters

1: **npde** – Integer *Input*

On entry: the number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **m** – Integer *Input*

On entry: the co-ordinate system used:

m $= 0$

Indicates Cartesian co-ordinates.

m $= 1$

Indicates cylindrical polar co-ordinates.

m $= 2$

Indicates spherical polar co-ordinates.

Constraint: $0 \leq$ **m** ≤ 2.

3: **ts** – double * *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**.

4: **tout** – double *Input*

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

5: **pdedef** *Function*

pdedef must evaluate the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. The functions may depend on x, t, U, U_x and V. Q_i may depend linearly on $\dot{V}$. **pdedef** is called approximately midway between each pair of mesh points in turn by nag_pde_parab_1d_fd_ode (d03phc).

```
void pdedef (Integer npde, double t, double x, const double u[],
     const double ux[], Integer ncode, const double v[], const double vdot[],
     double p[], double q[], double r[], Integer *ires, Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *Input*

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

3: **x** – double *Input*

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

4: **u**[**npde**] – const double *Input*

On entry: **u**$[i-1]$ contains the value of the component $U_i(x,t)$, for $i = 1, 2, \ldots,$ **npde**.

5: **ux**[**npde**] – const double *Input*

On entry: **ux**$[i-1]$ contains the value of the component $(\partial U_i(x,t))/(\partial x)$, for $i = 1, 2, \ldots,$ **npde**.

6: **ncode** – Integer *Input*

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

7: **v**[**ncode**] – const double *Input*

On entry: **v**$[i-1]$ contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

8: **vdot**[**ncode**] – const double *Input*

On entry: **vdot**$[i-1]$ contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

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

9: **p**[**npde** $\times$ **npde**] – double *Output*

Note: where $\mathbf{P}(i,j)$ appears in this document it refers to the array element **p**[**npde** $\times (j-1) + i - 1$]. We recommend using a #define to make the same definition in your calling program.

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

10: **q**[**npde**] – double *Output*

On exit: **q**$[i-1]$ must be set to the value of $Q_i(x,t,U,U_x,V,\dot{V})$, for $i = 1, 2, \ldots,$ **npde**.

11: **r**[**npde**] – double *Output*

On exit: **r**$[i-1]$ must be set to the value of $R_i(x,t,U,U_x,V)$, for $i = 1, 2, \ldots,$ **npde**.

12: **ires** – Integer * *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, the user 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 **fail.code** = **NE_USER_STOP**.

ires $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. The user may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If the user consecutively sets **ires** $= 3$, then nag_pde_parab_1d_fd_ode (d03phc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

13: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

6: **bndary** *Function*

bndary must evaluate the functions β_i and γ_i which describe the boundary conditions, as given in (4).

```
void bndary (Integer npde, double t, const double u[], const double ux[],
     Integer ncode, const double v[], const double vdot[], Integer ibnd,
     double beta[], double gamma[], Integer *ires, Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *Input*

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

3: **u**[**npde**] – const double *Input*

On entry: **u**$[i-1]$ contains the value of the component $U_i(x,t)$ at the boundary specified by **ibnd**, for $i = 1, 2, \dots,$ **npde**.

4: **ux**[**npde**] – const double *Input*

On entry: **ux**$[i-1]$ contains the value of the component $(\partial U_i(x,t))/(\partial x)$ at the boundary specified by **ibnd**, for $i = 1, 2, \dots,$ **npde**.

5: **ncode** – Integer *Input*

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

6: **v**[**ncode**] – const double *Input*

On entry: **v**$[i-1]$ contains the value of component $V_i(t)$, for $i = 1, 2, \dots,$ **ncode**.

7: **vdot**[**ncode**] – const double *Input*

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

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

8: **ibnd** – Integer *Input*

On entry: specifies which boundary conditions are to be evaluated. If **ibnd** $= 0$, then **bndary** must set up the coefficients of the left-hand boundary, $x = a$. If **ibnd** $\neq 0$, then **bndary** must set up the coefficients of the right-hand boundary, $x = b$.

9: **beta**[**npde**] – double *Output*

On exit: **beta**$[i-1]$ must be set to the value of $\beta_i(x,t)$ at the boundary specified by **ibnd**, for $i = 1, 2, \dots,$ **npde**.

10: **gamma**[**npde**] – double *Output*

On exit: **gamma**$[i-1]$ must be set to the value of $\gamma_i(x,t,U,U_x,V,\dot{V})$ at the boundary specified by **ibnd**, for $i = 1, 2, \dots,$ **npde**.

11: **ires** – Integer * *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, the user 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 **fail.code** = **NE_USER_STOP**.

ires $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. The user may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If the user consecutively sets **ires** $= 3$, then nag_pde_parab_1d_fd_ode (d03phc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

12: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

7: **u**[**neqn**] – double *Input/Output*

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

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

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

8: **npts** – Integer *Input*

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

Constraint: **npts** ≥ 3.

9: **x**[**npts**] – const double *Input*

On entry: the mesh points in the space direction. **x**[0] must specify the left-hand boundary, a, and **x**[**npts** -1] must specify the right-hand boundary, b.

Constraint: **x**[0] $<$ **x**[1] $< \cdots <$ **x**[**npts** -1].

10: **ncode** – Integer *Input*

On entry: the number of coupled ODE components.

Constraint: **ncode** ≥ 0.

11: **odedef** *Function*

odedef must evaluate the functions F, which define the system of ODEs, as given in (3). If the user wishes to compute the solution of a system of PDEs only (**ncode** $= 0$), **odedef** must be the dummy function d03pck. (d03pck is included in the NAG C Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details.)

```
void odedef (Integer npde, double t, Integer ncode, const double v[],
     const double vdot[], Integer nxi, const double xi[], const double ucp[],
     const double ucpx[], const double rcp[], const double ucpt[],
     const double ucptx[], double f[], Integer *ires, Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *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**] – const double *Input*

On entry: **v**$[i-1]$ contains the value of component $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

5: **vdot**[**ncode**] – const double *Input*

On entry: **vdot**$[i-1]$ 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**] – const double *Input*

On entry: **xi**$[i-1]$ contains the ODE/PDE coupling points, ξ_i, $i = 1, 2, \ldots,$ **nxi**.

8: **ucp**[**npde** $\times$ **nxi**] – const double *Input*

Note: where **UCP**(i,j) appears in this document it refers to the array element **ucp**[**npde** $\times (j-1)+i-1$]. We recommend using a #define to make the same definition in your calling program.

On entry: **UCP**(i,j) contains the value of $U_i(x,t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

9: **ucpx**[**npde** $\times$ **nxi**] – const double *Input*

Note: where **UCPX**(i,j) appears in this document it refers to the array element **ucpx**[**npde** $\times (j-1)+i-1$]. We recommend using a #define to make the same definition in your calling program.

On entry: **UCPX**(i,j) contains the value of $(\partial U_i(x,t))/(\partial x)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

10: **rcp**[**npde** $\times$ **nxi**] – const double *Input*

Note: where **RCP**(i,j) appears in this document it refers to the array element **rcp**[**npde** $\times (j-1)+i-1$]. We recommend using a #define to make the same definition in your calling program.

On entry: **RCP**(i,j) contains the value of the flux R_i at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

11: **ucpt**[**npde** $\times$ **nxi**] – const double *Input*

Note: where **UCPT**(i,j) appears in this document it refers to the array element **ucpt**[**npde** $\times (j-1)+i-1$]. We recommend using a #define to make the same definition in your calling program.

On entry: **UCPT**(i,j) contains the value of $(\partial U_i)/(\partial t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

12: **ucptx**[**npde** $\times$ **nxi**] – const double *Input*

Note: where **UCPTX**(i,j) appears in this document it refers to the array element

ucptx[**npde** $\times (j - 1) + i - 1$]. We recommend using a #define to make the same definition in your calling program.

On entry: **UCPTX**(i, j) contains the value of $(\partial^2 U_i)/(\partial x \partial t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

13: **f**[**ncode**] – double *Output*

On exit: **f**$[i - 1]$ must contain the ith component of F, for $i = 1, 2, \ldots,$ **ncode**, where F is defined as

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{5}$$

or

$$F = -A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}. \tag{6}$$

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

14: **ires** – Integer * *Input/Output*

On entry: the form of F that must be returned in the array **f**. If **ires** $= 1$, then the equation (5) above must be used. If **ires** $= -1$, then the equation (6) above must be used.

On exit: should usually remain unchanged. However, the user may reset **ires** to force the integration function to take certain actions as described below:

ires $= 2$

Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to **fail.code** = **NE_USER_STOP**.

ires $= 3$

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. The user may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If the user consecutively sets **ires** $= 3$, then nag_pde_parab_1d_fd_ode (d03phc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

15: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

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**[dim] – const double *Input*

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

On entry: **xi**$[i - 1]$, $i = 1, 2, \ldots,$ **nxi**, must be set to the ODE/PDE coupling points.

Constraint: **x**$[0] \leq$ **xi**$[0] <$ **xi**$[1] < \cdots <$ **xi**$[$**nxi** $- 1] \leq$ **x**$[$**npts** $- 1]$.

14: **neqn** – Integer *Input*

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

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

15: **rtol**[dim] – const double *Input*

Note: the dimension, dim, of the array **rtol** must be at least 1 when **itol** $= 1$ or 2 and at least **neqn** when **itol** $= 3$ or 4.

On entry: the relative local error tolerance.

Constraint: **rtol**$[i-1] \geq 0$ for all relevant i.

16: **atol**[dim] – const double *Input*

Note: the dimension, dim, of the array **atol** must be at least 1 when **itol** $= 1$ or 3 and at least **neqn** when **itol** $= 2$ or 4.

On entry: the absolute local error tolerance.

Constraint: **atol**$[i-1] \geq 0$ for all relevant i.

17: **itol** – Integer *Input*

On entry: a value to indicate the form of the local error test. **itol** indicates to nag_pde_parab_1d_fd_ode (d03phc) whether to interpret either or both of **rtol** or **atol** as a vector or scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

itol	**rtol**	**atol**	w_i
1	scalar	scalar	**rtol**$[0] \times \lvert U_i \rvert +$ **atol**$[0]$
2	scalar	vector	**rtol**$[0] \times \lvert U_i \rvert +$ **atol**$[i-1]$
3	vector	scalar	**rtol**$[i-1] \times \lvert U_i \rvert +$ **atol**$[0]$
4	vector	vector	**rtol**$[i-1] \times \lvert U_i \rvert +$ **atol**$[i-1]$

In the above, e_i denotes the estimated local error for the ith component of the coupled PDE/ODE system in time, **u**$[i-1]$, for $i = 1, 2, \ldots,$ **neqn**.

The choice of norm used is defined by the parameter **norm**, see below.

Constraint: $1 \leq$ **itol** ≤ 4.

18: **norm** – Nag_NormType *Input*

On entry: the type of norm to be used. Two options are available:

norm = **Nag_MaxNorm**

Maximum norm.

norm = **Nag_TwoNorm**

Averaged L_2 norm.

If $\mathbf{u}_{norm}$ denotes the norm of the vector **u** of length **neqn**, then for the averaged L_2 norm

$$\mathbf{u}_{norm} = \sqrt{\frac{1}{\mathbf{neqn}} \sum_{i=1}^{\mathbf{neqn}} (\mathbf{u}[i-1]/w_i)^2},$$

while for the maximum norm

$$\mathbf{u}_{norm} = \max_i |\mathbf{u}[i-1]/w_i|.$$

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

Constraint: **norm** = **Nag_MaxNorm** or **Nag_TwoNorm**.

19: **laopt** – Nag_LinAlgOption *Input*

On entry: the type of matrix algebra required. The possible choices are:

laopt = **Nag_LinAlgFull**

Full matrix methods to be used.

laopt = **Nag_LinAlgBand**

Banded matrix methods to be used.

laopt = **Nag_LinAlgSparse**

Sparse matrix methods to be used.

Constraint: **laopt** = **Nag_LinAlgFull**, **Nag_LinAlgBand** or **Nag_LinAlgSparse**.

Note: the user is recommended to use the banded option when no coupled ODEs are present (i.e., **ncode** = 0).

20: **algopt**[30] – const double *Input*

On entry: **algopt** may be set to control various options available in the integrator. If the user wishes to employ all the default options, then **algopt**[0] 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[0] selects the ODE integration method to be used. If **algopt**[0] = 1.0, a BDF method is used and if **algopt**[0] = 2.0, a Theta method is used. The default value is **algopt**[0] = 1.0.

If **algopt**[0] = 2.0, then **algopt**[i], for $i = 1, 2, 3$ are not used.

algopt[1] specifies the maximum order of the BDF integration formula to be used. **algopt**[1] may be 1.0, 2.0, 3.0, 4.0 or 5.0. The default value is **algopt**[1] = 5.0.

algopt[2] specifies what method is to be used to solve the system of nonlinear equations arising on each step of the BDF method. If **algopt**[2] = 1.0 a modified Newton iteration is used and if **algopt**[2] = 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**[2] = 1.0.

algopt[3] 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**[3] = 1.0, then the Petzold test is used. If **algopt**[3] = 2.0, then the Petzold test is not used. The default value is **algopt**[3] = 1.0.

If **algopt**[0] = 1.0, then **algopt**[i], for $i = 4, 5, 6$ are not used.

algopt[4] specifies the value of Theta to be used in the Theta integration method. $0.51 \leq$ **algopt**[4] ≤ 0.99. The default value is **algopt**[4] = 0.55.

algopt[5] specifies what method is to be used to solve the system of nonlinear equations arising on each step of the Theta method. If **algopt**[5] = 1.0, a modified Newton iteration is used and if **algopt**[5] = 2.0, a functional iteration method is used. The default value is **algopt**[5] = 1.0.

algopt[6] 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**[6] = 1.0, then switching is allowed and if **algopt**[6] = 2.0, then switching is not allowed. The default value is **algopt**[6] = 1.0.

algopt[10] 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**[0] $\neq 0.0$, a value of 0.0 for **algopt**[10], say, should be specified even if **itask** subsequently specifies that t_{crit} will not be used.

algopt[11] specifies the minimum absolute step size to be allowed in the time integration. If this option is not required, **algopt**[11] should be set to 0.0.

algopt[12] specifies the maximum 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 initial step size to be attempted by the integrator. If **algopt**$[13] = 0.0$, then the initial step size is calculated internally.

algopt[14] specifies the maximum number of steps to be attempted by the integrator in any one call. If **algopt**$[14] = 0.0$, then no limit is imposed.

algopt[22] specifies what method is to be used to solve the nonlinear equations at the initial point to initialise the values of U, U_t, V and $\dot{V}$. If **algopt**$[22] = 1.0$, a modified Newton iteration is used and if **algopt**$[22] = 2.0$, functional iteration is used. The default value is **algopt**$[22] = 1.0$.

algopt[28] and **algopt**[29] are used only for the sparse matrix algebra option, **laopt** = **Nag_LinAlgSparse**.

algopt[28] governs the choice of pivots during the decomposition of the first Jacobian matrix. It should lie in the range $0.0 <$ **algopt**$[28] < 1.0$, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If **algopt**[28] lies outside this range then the default value is used. If the functions regard the Jacobian matrix as numerically singular then increasing **algopt**[28] towards 1.0 may help, but at the cost of increased fill-in. The default value is **algopt**$[28] = 0.1$.

algopt[29] is used as a relative pivot threshold during subsequent Jacobian decompositions (see **algopt**[28]) below which an internal error is invoked. If **algopt**[29] is greater than 1.0 no check is made on the pivot size, and this may be a necessary option if the Jacobian is found to be numerically singular (see **algopt**[28]). The default value is **algopt**$[29] = 0.0001$.

21: **rsave**[**lrsave**] – double *Input/Output*

On entry: if **ind** $= 0$, **rsave** need not be set. If **ind** $= 1$ then it must be unchanged from the previous call to the function.

On exit: contains information about the iteration required for subsequent calls.

22: **lrsave** – Integer *Input*

On entry: the dimension of the array **rsave** as declared in the function from which nag_pde_parab_1d_fd_ode (d03phc) is called.

Its size depends on the type of matrix algebra selected:

if **laopt** = **Nag_LinAlgFull**, **lrsave** $\geq$ **neqn** $\times$ **neqn** $+$ **neqn** $+ nwkres + lenode$;

if **laopt** = **Nag_LinAlgBand**, **lrsave** $\geq (3 \times mlu + 1) \times$ **neqn** $+ nwkres + lenode$;

if **laopt** = **Nag_LinAlgSparse**, **lrsave** $\geq 4 \times$ **neqn** $+ 11 \times$ **neqn**$/2 + 1 + nwkres + lenode$,

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

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

$lenode = (6 + \mathrm{int}($**algopt**$[1])) \times$ **neqn** $+ 50$, when the BDF method is used, and
$lenode = 9 \times$ **neqn** $+ 50$, when the Theta method is used.

Note: when **laopt** $=$ **Nag_LinAlgSparse**, the value of **lrsave** may be too small when supplied to the integrator. An estimate of the minimum size of **lrsave** is printed on the current error message unit if **itrace** > 0 and the function returns with **fail.code** $=$ **NE_INT_2**.

23: **isave**[**lisave**] – Integer *Input/Output*

On entry: if **ind** $= 0$, **isave** need not be set. If **ind** $= 1$ then it must be unchanged from the previous call to the function.

On exit: contains information about the iteration required for subsequent calls. In particular:

isave[0] contains the number of steps taken in time.

isave[1] 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[2] contains the number of Jacobian evaluations performed by the time integrator.

isave[3] contains the order of the last backward differentiation formula method used.

isave[4] 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.

24: **lisave** – Integer *Input*

On entry: the dimension of the array **isave** as declared in the function from which nag_pde_parab_1d_fd_ode (d03phc) is called. Its size depends on the type of matrix algebra selected:

if **laopt** $=$ **Nag_LinAlgFull**, **lisave** ≥ 24

if **laopt** $=$ **Nag_LinAlgBand**, **lisave** $\geq$ **neqn** $+ 24$

if **laopt** $=$ **Nag_LinAlgSparse**, **lisave** $\geq 25 \times$ **neqn** $+ 24$

Note: when using the sparse option, the value of **lisave** may be too small when supplied to the integrator. An estimate of the minimum size of **lisave** is printed if **itrace** > 0 and the function returns with **fail.code** $=$ **NE_INT_2**.

25: **itask** – Integer *Input*

On entry: specifies the task to be performed by the ODE integrator. The permitted values of **itask** and their meanings are detailed below:

itask $= 1$

Normal computation of output values **u** at $t =$ **tout**.

itask $= 2$

One step and return.

itask $= 3$

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

itask $= 4$

Normal computation of output values **u** at $t =$ **tout** but without overshooting $t = t_{\text{crit}}$ where t_{crit} is described under the parameter **algopt**.

itask $= 5$

Take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter **algopt**.

Constraint: $1 \leq$ **itask** ≤ 5.

26: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_pde_parab_1d_fd_ode (d03phc) and the underlying ODE solver. **itrace** may take the value -1, 0, 1, 2, or 3. If **itrace** < -1, then -1 is assumed and similarly if **itrace** > 3, then 3 is assumed. If **itrace** $= -1$, no output is generated. If **itrace** $= 0$, only warning messages from the PDE solver are printed. If **itrace** > 0, then output from the underlying ODE solver is printed. This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system. The advisory messages are given in greater detail as **itrace** increases.

27: **outfile** – char * *Input*

On entry: the name of a file to which diagnostic output will be directed. If **outfile** is NULL the diagnostic output will be directed to standard output.

28: **ind** – Integer * *Input/Output*

On entry: **ind** must be set to 0 or 1.

ind $= 0$

Starts or restarts the integration in time.

ind $= 1$

Continues the integration after an earlier exit from the function. In this case, only the parameters **tout** and **fail** should be reset between calls to nag_pde_parab_1d_fd_ode (d03phc).

Constraint: $0 \leq$ **ind** ≤ 1.

On exit: **ind** $= 1$.

29: **comm** – NAG_Comm * *Input/Output*

The NAG communication parameter (see the Essential Introduction).

30: **saved** – Nag_D03_Save * *Input/Output*

Note: saved is a NAG defined structure. See Section 2.2.1.1 of the Essential Introduction.

On entry: if the current call to nag_pde_parab_1d_fd_ode (d03phc) follows a previous call to a Chapter d03 function then **saved** must contain the unchanged value output from that previous call.

On exit: data to be passed unchanged to any subsequent call to a Chapter d03 function.

31: **fail** – NagError * *Input/Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **npde** = ⟨*value*⟩.
Constraint: **npde** ≥ 1.

On entry, **npts** = ⟨*value*⟩.
Constraint: **npts** ≥ 3.

On entry, **ncode** = ⟨*value*⟩.
Constraint: **ncode** ≥ 0.

ires set to an invalid value in call to **pdedef**, **bndary**, or **odedef**.

On entry, **m** is not equal to 0, 1, or 2: **m** = ⟨*value*⟩.

On entry, **itol** is not equal to 1, 2, 3, or 4: **itol** = ⟨*value*⟩.

On entry, **ind** is not equal to 0 or 1: **ind** $= \langle value \rangle$.

On entry, **itask** is not equal to 1, 2, 3, 4 or 5: **itask** $= \langle value \rangle$.

On entry, **ncode** $= 0$, but **nxi** is not equal to 0: **nxi** $= \langle value \rangle$.

On entry, **nxi** $= \langle value \rangle$.
Constraint: **nxi** ≥ 0.

NE_INT_2

On entry, **ncode** $= \langle value \rangle$, **nxi** $= \langle value \rangle$.
Constraint: if **ncode** $= 0$, **nxi** $= 0$;
if **ncode** > 0, **nxi** ≥ 0.

When using the sparse option **lisave** or **lrsave** is too small: **lisave** $= \langle value \rangle$, **lrsave** $= \langle value \rangle$.

On entry, corresponding elements **atol**$[i-1]$ and **rtol**$[j-1]$ are both zero. $i = \langle value \rangle$, $j = \langle value \rangle$.

On entry, **lrsave** is too small: **lrsave** $= \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$.

On entry, **lisave** is too small: **lisave** $= \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$.

NE_INT_4

On entry, **neqn** $= \langle value \rangle$, **npde** $= \langle value \rangle$, **npts** $= \langle value \rangle$, **ncode** $= \langle value \rangle$.
Constraint: **neqn** $=$ **npde** $\times$ **npts** $+$ **ncode**.

On entry, **neqn** is not equal to **npde** $\times$ **npts** $+$ **ncode**: **neqn** $= \langle value \rangle$, **npde** $= \langle value \rangle$, **npts** $= \langle value \rangle$, **ncode** $= \langle value \rangle$.

NE_ACC_IN_DOUBT

Integration completed, but small changes in **atol** or **rtol** are unlikely to result in a changed solution.

NE_FAILED_DERIV

In setting up the ODE system an internal auxiliary was unable to initialize the derivative. This could be due to user setting **ires** $= 3$ in **pdedef** or **bndary**.

NE_FAILED_START

atol and **rtol** were too small to start integration.

NE_FAILED_STEP

Repeated errors in an attempted step of underlying ODE solver. Integration was successful as far as **ts**: **ts** $= \langle value \rangle$.

Error during Jacobian formulation for ODE system. Increase **itrace** for further details.

Underlying ODE solver cannot make further progress from the point **ts** with the supplied values of **atol** and **rtol**. **ts** $= \langle value \rangle$.

NE_INCOMPAT_PARAM

On entry, **m** > 0 and **x**$[0] < 0.0$: **m** $= \langle value \rangle$, **x**$[0] = \langle value \rangle$.

NE_INTERNAL_ERROR

Serious error in internal call to an auxiliary. Increase **itrace** for further details.

NE_ITER_FAIL

In solving ODE system, the maximum number of steps **algopt**$[14]$ has been exceeded. **algopt**$[14] = \langle value \rangle$.

NE_NOT_STRICTLY_INCREASING

On entry $\mathbf{xi}[i] \leq \mathbf{xi}[i-1]$: $i = \langle value \rangle$, $\mathbf{xi}[i] = \langle value \rangle$, $\mathbf{xi}[i-1] = \langle value \rangle$.

On entry, mesh points **x** appear to be badly ordered: $i = \langle value \rangle$, $\mathbf{x}[i-1] = \langle value \rangle$ $j = \langle value \rangle$, $\mathbf{x}[j-1] = \langle value \rangle$.

NE_REAL_2

On entry, at least one point in **xi** lies outside $[\mathbf{x}[0], \mathbf{x}[\mathbf{npts}-1]]$: $\mathbf{x}[0] = \langle value \rangle$, $\mathbf{x}[\mathbf{npts}-1] = \langle value \rangle$.

On entry, $\mathbf{tout} - \mathbf{ts}$ is too small: $\mathbf{tout} = \langle value \rangle$, $\mathbf{ts} = \langle value \rangle$.

On entry, $\mathbf{tout} \leq \mathbf{ts}$: $\mathbf{tout} = \langle value \rangle$, $\mathbf{ts} = \langle value \rangle$.

NE_REAL_ARRAY

On entry, $\mathbf{rtol}[i-1] < 0.0$: $i = \langle value \rangle$, $\mathbf{rtol}[i-1] = \langle value \rangle$.

On entry, $\mathbf{atol}[i-1] < 0.0$: $i = \langle value \rangle$, $\mathbf{atol}[i-1] = \langle value \rangle$.

NE_SING_JAC

Singular Jacobian of ODE system. Check problem formulation.

NE_TIME_DERIV_DEP

Flux function appears to depend on time derivatives.

NE_USER_STOP

In evaluating residual of ODE system, $\mathbf{ires} = 2$ has been set in **pdedef**, **bndary**, or **odedef**. Integration is successful as far as **ts**: $\mathbf{ts} = \langle value \rangle$.

NE_ZERO_WTS

Zero error weights encountered during time integration.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ had an illegal value.

NE_NOT_WRITE_FILE

Cannot open file $\langle value \rangle$ for writing.

NE_NOT_CLOSE_FILE

Cannot close file $\langle value \rangle$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

7 Accuracy

The function 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. The user should therefore test the effect of varying the accuracy parameters **atol** and **rtol**.

8 Further Comments

The parameter specification allows the user to include equations with only first-order derivatives in the space direction but there is no guarantee that the method of integration will be satisfactory for such systems. The position and nature of the boundary conditions in particular are critical in defining a stable problem. It may be advisable in such cases to reduce the whole system to first-order and to use the Keller box scheme function nag_pde_parab_1d_keller_ode (d03pkc).

The time taken depends on the complexity of the parabolic system and on the accuracy requested. For a given system and a fixed accuracy it is approximately proportional to **neqn**.

9 Example

This problem provides a simple coupled system of one PDE and one ODE.

$$(V_1)^2 \frac{\partial U_1}{\partial t} - xV_1\dot{V}_1\frac{\partial U_1}{\partial x} = \frac{\partial^2 U_1}{\partial x^2}$$

$$\dot{V}_1 = V_1 U_1 + \frac{\partial U_1}{\partial x} + 1 + t,$$

for $t \in [10^{-4}, 0.1 \times 2^i]$, for $i = 1, 2, \ldots, 5,\ x \in [0, 1]$.

The left boundary condition at $x = 0$ is

$$\frac{\partial U_1}{\partial x} = -V_1 \exp t.$$

The right boundary condition at $x = 1$ is

$$\frac{\partial U_1}{\partial x} = -V_1\dot{V}_1$$

The initial conditions at $t = 10^{-4}$ are defined by the exact solution:

$$V_1 = t, \quad \text{and} \quad U_1(x,t) = \exp\{t(1-x)\} - 1.0,\ x \in [0,1],$$

and the coupling point is at $\xi_1 = 1.0$.

9.1 Program Text

```
/* nag_pde_parab_1d_fd_ode (d03phc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd03.h>

static void pdedef(Integer, double, double, const double[], const double[],
                   Integer, const double[], const double[], double[],
                   double[], double[], Integer *, Nag_Comm *);

static void bndary(Integer, double, const double[], const double[],
                   Integer, const double[], const double[], Integer,
                   double[], double[], Integer *, Nag_Comm *);

static void odedef(Integer, double, Integer, const double[], const double[],
                   Integer, const double[], const double[], const double[],
```

```
                   const double[], const double[], const double[], double[],
                   Integer *, Nag_Comm *);

static void uvinit(Integer, Integer, double *, double *, Integer, Integer,
                   double);

static void exact(double, Integer, double *, double *);

#define P(I,J) p[npde*((J)-1)+(I)-1]
#define UCPX(I,J) ucpx[npde*((J)-1)+(I)-1]
#define UCP(I,J) ucp[npde*((J)-1)+(I)-1]

int main(void)
{
  const Integer npde=1, npts=21, ncode=1, m=0, nxi=1,
    neqn=npde*npts+ncode, lisave=24, lenode=11*neqn+50,
    nwkres=npde*(npts+6*nxi+3*npde+15)+ncode+nxi+7*npts+2,
    lrsave=neqn*neqn+neqn+nwkres+lenode;
  double tout, ts;
  Integer exit_status, i, ind, it, itask, itol, itrace;
  Boolean theta;
  double *algopt=0, *atol=0, *exy=0, *rsave=0, *rtol=0,
    *u=0, *x=0, *xi=0;
  Integer *isave=0;
  NagError fail;
  Nag_Comm comm;
  Nag_D03_Save saved;

  /* Allocate memory */

  if ( !(algopt = NAG_ALLOC(30, double)) ||
       !(atol = NAG_ALLOC(1, double)) ||
       !(exy = NAG_ALLOC(npts, double)) ||
       !(rsave = NAG_ALLOC(lrsave, double)) ||
       !(rtol = NAG_ALLOC(1, double)) ||
       !(u = NAG_ALLOC(neqn, double)) ||
       !(x = NAG_ALLOC(npts, double)) ||
       !(xi = NAG_ALLOC(1, double)) ||
       !(isave = NAG_ALLOC(lisave, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = 1;
      goto END;
    }

  Vprintf("d03phc Example Program Results\n\n");
  INIT_FAIL(fail);
  exit_status = 0;

  itrace = 0;
  itol = 1;
  atol[0] = 1e-4;
  rtol[0] = atol[0];

  Vprintf("  Simple coupled PDE using BDF\n");
  Vprintf("  Accuracy requirement =%10.3e", atol[0]);
  Vprintf(" Number of points = %4ld\n\n", npts);

  /* Set break-points */

  for (i = 0; i < npts; ++i)
    {
      x[i] = i/(npts-1.0);
    }

  xi[0] = 1.0;
  ind = 0;
  itask = 1;

  /* Set theta = TRUE if the Theta integrator is required */
```

```
  theta = FALSE;
  for (i = 0; i < 30; ++i) algopt[i] = 0.0;
  if (theta) algopt[0] = 2.0;

  /* Loop over output value of t */

  ts = 1e-4;
  tout = 0.0;
  Vprintf("  x        %9.3f%9.3f%9.3f%9.3f%9.3f\n\n",
          x[0], x[4], x[8], x[12], x[20]);
  uvinit(npde, npts, x, u, npde, neqn, ts);

  for (it = 0; it < 5; ++it)
    {
      tout = 0.1*pow(2.0, (it+1));

      d03phc(npde, m, &ts, tout, pdedef, bndary, u, npts, x, ncode,
             odedef, nxi, xi, neqn, rtol, atol, itol, Nag_TwoNorm,
             Nag_LinAlgFull, algopt, rsave, lrsave, isave, lisave,
             itask, itrace, 0, &ind, &comm, &saved, &fail);

      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from d03phc.\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Check against the exact solution */

      exact(tout, npts, x, exy);
      Vprintf(" t = %6.3f\n", ts);
      Vprintf(" App.  sol.  %7.3f%9.3f%9.3f%9.3f%9.3f",
              u[0], u[4], u[8], u[12], u[20]);
      Vprintf("  ODE sol. =%8.3f\n", u[21]);
      Vprintf(" Exact sol.  %7.3f%9.3f%9.3f%9.3f%9.3f",
              exy[0], exy[4], exy[8], exy[12], exy[20]);
      Vprintf("  ODE sol. =%8.3f\n\n", ts);
    }
  Vprintf(" Number of integration steps in time = %6ld\n", isave[0]);
  Vprintf(" Number of function evaluations = %6ld\n", isave[1]);
  Vprintf(" Number of Jacobian evaluations =%6ld\n", isave[2]);
  Vprintf(" Number of iterations = %6ld\n\n", isave[4]);
 END:
  if (algopt) NAG_FREE(algopt);
  if (atol) NAG_FREE(atol);
  if (exy) NAG_FREE(exy);
  if (rsave) NAG_FREE(rsave);
  if (rtol) NAG_FREE(rtol);
  if (u) NAG_FREE(u);
  if (x) NAG_FREE(x);
  if (xi) NAG_FREE(xi);
  if (isave) NAG_FREE(isave);

  return exit_status;
}

static void pdedef(Integer npde, double t, double x, const double u[],
                   const double ux[], Integer ncode, const double v[],
                   const double vdot[], double p[], double q[],
                   double r[], Integer *ires, Nag_Comm *comm)
{
  P(1, 1) = v[0] * v[0];
  r[0] = ux[0];
  q[0] = -(x) * ux[0] * v[0] * vdot[0];

  return;
}

static void bndary(Integer npde, double t, const double u[],
                   const double ux[], Integer ncode, const double v[],
```

```
                   const double vdot[], Integer ibnd, double beta[],
                   double gamma[], Integer *ires, Nag_Comm *comm)
{
  beta[0] = 1.0;
  if (ibnd == 0)
    {
      gamma[0] = -v[0]*exp(t);
    } else {
      gamma[0] = -v[0]*vdot[0];
    }
  return;
}

static void odedef(Integer npde, double t, Integer ncode, const double v[],
                   const double vdot[], Integer nxi, const double xi[],
                   const double ucp[], const double ucpx[],
                   const double rcp[], const double ucpt[],
                   const double ucptx[], double f[], Integer *ires,
                   Nag_Comm *comm)
{
  if (*ires == 1)
    {
      f[0] = vdot[0] - v[0] * UCP(1, 1) - UCPX(1, 1) - 1.0 - t;
    } else if (*ires == -1) {
      f[0] = vdot[0];
    }
  return;
}

static void uvinit(Integer npde, Integer npts, double *x,
            double *u, Integer ncode, Integer neqn, double ts)
{

  /* Routine for PDE initial values */

  Integer i;

  for (i = 0; i < npts; ++i)
    {
      u[i] = exp(ts*(1.0 - x[i])) - 1.0;
    }
  u[neqn-1] = ts;

  return;
}

static void exact(double time, Integer npts, double *x, double *u)
{

  /* Exact solution (for comparison purpose) */

  Integer i;
  for (i = 0; i < npts; ++i)
    {
      u[i] = exp(time*(1.0 - x[i])) - 1.0;
    }
  return;
}
```

9.2 Program Data

None.

9.3 Program Results

```
d03phc Example Program Results

  Simple coupled PDE using BDF
  Accuracy requirement = 1.000e-04 Number of points =   21
```

```
 x           0.000    0.200    0.400    0.600    1.000

t =  0.200
App.  sol.   0.222    0.174    0.128    0.084    0.001  ODE sol. =   0.200
Exact sol.   0.221    0.174    0.127    0.083    0.000  ODE sol. =   0.200

t =  0.400
App.  sol.   0.494    0.379    0.273    0.176    0.002  ODE sol. =   0.400
Exact sol.   0.492    0.377    0.271    0.174    0.000  ODE sol. =   0.400

t =  0.800
App.  sol.   1.229    0.901    0.622    0.384    0.008  ODE sol. =   0.798
Exact sol.   1.226    0.896    0.616    0.377    0.000  ODE sol. =   0.800

t =  1.600
App.  sol.   3.959    2.610    1.629    0.917    0.027  ODE sol. =   1.594
Exact sol.   3.953    2.597    1.612    0.896    0.000  ODE sol. =   1.600

t =  3.200
App.  sol.  23.469   11.974    5.885    2.665    0.074  ODE sol. =   3.184
Exact sol.  23.533   11.936    5.821    2.597    0.000  ODE sol. =   3.200

Number of integration steps in time =     33
Number of function evaluations =    470
Number of Jacobian evaluations =    16
Number of iterations =    111
```