NAG Toolbox

nag_pde_1d_parab_dae_keller (d03pk)

1 Purpose

nag_pde_1d_parab_dae_keller (d03pk) integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs). The spatial discretization is performed using the Keller box scheme 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 (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Syntax

```
[ts, u, rsave, isave, ind, ifail] = nag_pde_1d_parab_dae_keller(npde, ts, tout, pdedef, bndary, u, x, nleft, ncode, odedef, xi, rtol, atol, itol, norm_p, laopt, algopt, rsave, isave, itask, itrace, ind, 'npts', npts, 'nxi', nxi, 'neqn', neqn)
[ts, u, rsave, isave, ind, ifail] = d03pk(npde, ts, tout, pdedef, bndary, u, x, nleft, ncode, odedef, xi, rtol, atol, itol, norm_p, laopt, algopt, rsave, isave, itask, itrace, ind, 'npts', npts, 'nxi', nxi, 'neqn', neqn)
```

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_dae_keller (d03pk) integrates the system of first-order PDEs and coupled ODEs

$$G_i\left(x,t,U,U_x,U_t,V,\dot{V}\right)=0, \quad i=1,2,\ldots,\mathbf{npde}, \quad a\le x\le b, t\ge t_0, \tag{1}$$

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

In the PDE part of the problem given by (1), the functions G_i must have the general form

$$G_i=\sum_{j=1}^{\mathbf{npde}}P_{i,j}\frac{\partial U_j}{\partial t}+\sum_{j=1}^{\mathbf{ncode}}Q_{i,j}\dot{V}_j+S_i=0, \quad i=1,2,\ldots,\mathbf{npde}, \tag{3}$$

where $P_{i,j}$, $Q_{i,j}$ and S_i depend on x,t,U,U_x and V.

The vector U is the set of PDE solution values

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

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

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

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

In the ODE part given by (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^* and U_t^* are the functions U, U_x and U_t evaluated at these coupling points. Each R_i may only depend linearly on time derivatives. Hence equation (2) may be written more precisely as

$$R=A-B\dot{V}-CU_t^*, \tag{4}$$

where $R = [R_1, \ldots, R_{\mathbf{ncode}}]^{\mathrm{T}}$, A is a vector of length **ncode**, B is an **ncode** by **ncode** matrix, C is an **ncode** by $\left(n_\xi \times \mathbf{npde}\right)$ matrix. The entries in A, B and C 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 B and C. (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 $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$.

The PDE system which is defined by the functions G_i must be specified in **pdedef**.

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

For a first-order system of PDEs, only one boundary condition is required for each PDE component U_i. The **npde** boundary conditions are separated into n_a at the left-hand boundary $x = a$, and n_b at the right-hand boundary $x = b$, such that $n_a + n_b = \mathbf{npde}$. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of U_i at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for U_i should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialization or integration difficulties in the underlying time integration functions.

The boundary conditions have the form:

$$G_i^L\left(x, t, U, U_t, V, \dot{V}\right) = 0 \quad \text{at } x = a, \quad i = 1, 2, \ldots, n_a, \tag{5}$$

at the left-hand boundary, and

$$G_i^R\left(x, t, U, U_t, V, \dot{V}\right) = 0 \quad \text{at } x = b, \quad i = 1, 2, \ldots, n_b, \tag{6}$$

at the right-hand boundary.

Note that the functions G_i^L and G_i^R must not depend on U_x, since spatial derivatives are not determined explicitly in the Keller box scheme. If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that G_i^L and G_i^R must be linear with respect to time derivatives, so that the boundary conditions have the general form:

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^L \dot{V}_j + K_i^L = 0, \quad i = 1, 2, \ldots, n_a, \tag{7}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^R \dot{V}_j + K_i^R = 0, \quad i = 1, 2, \ldots, n_b, \tag{8}$$

at the right-hand boundary, where $E_{i,j}^L$, $E_{i,j}^R$, $H_{i,j}^L$, $H_{i,j}^R$, K_i^L and K_i^R depend on x, t, U and V only.

The boundary conditions must be specified in **bndary**.

The problem is subject to the following restrictions:

(i) $P_{i,j}$, $Q_{i,j}$ and S_i must not depend on any time derivatives;

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

(iii) The evaluation of the function G_i is done approximately at the mid-points of the mesh $\mathbf{x}(i)$, for $i = 1, 2, \ldots,$ **npts**, by calling the **pdedef** for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathbf{npts}}$;

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

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 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. In this method of lines approach the Keller box scheme (see Keller (1970)) is applied to each

PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are **npde** $\times$ **npts** $+$ **ncode** ODEs in 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* (eds 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

Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** 327–350 Academic Press

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

5 Parameters

5.1 Compulsory Input Parameters

1: **npde** – INTEGER

The number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **ts** – REAL (KIND=nag_wp)

The initial value of the independent variable t.

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

3: **tout** – REAL (KIND=nag_wp)

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

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

pdedef must evaluate the functions G_i which define the system of PDEs. **pdedef** is called approximately midway between each pair of mesh points in turn by nag_pde_1d_parab_dae_keller (d03pk).

```
[res, ires] = pdedef(npde, t, x, u, ut, ux, ncode, v, vdot, 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: **ut**(**npde**) – REAL (KIND=nag_wp) array

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

6: **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**.

7: **ncode** – INTEGER

The number of coupled ODEs in the system.

8: **v**(**ncode**) – REAL (KIND=nag_wp) array

If **ncode** > 0, **v**(i) contains the value of the component $V_i(t)$, for $i = 1,2,\ldots,$ **ncode**.

9: **vdot**(**ncode**) – REAL (KIND=nag_wp) array

If **ncode** > 0, **vdot**(i) contains the value of component $\dot{V}_i(t)$, for $i = 1,2,\ldots,$ **ncode**.

10: **ires** – INTEGER

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

ires $= -1$
Equation (9) must be used.

ires $= 1$
Equation (10) must be used.

Output Parameters

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

res(i) must contain the ith component of G, for $i = 1,2,\ldots,$ **npde**, where G is defined as

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} Q_{i,j}\dot{V}_j, \tag{9}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} Q_{i,j}\dot{V}_j + S_i, \tag{10}$$

i.e., all terms in equation (3).

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

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 (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 nag_pde_1d_parab_dae_keller (d03pk) 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 boundary conditions, as given in (5) and (6).

```
[res, ires] = bndary(npde, t, ibnd, nobc, u, ut, ncode, v, vdot, 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: **ibnd** – INTEGER

Specifies which boundary conditions are to be evaluated.

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

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

4: **nobc** – INTEGER

Specifies the number of boundary conditions at the boundary specified by **ibnd**.

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

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

6: **ut**(**npde**) – REAL (KIND=nag_wp) array

ut(i) contains the value of the component $\dfrac{\partial U_i(x,t)}{\partial t}$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

7: **ncode** – INTEGER

The number of coupled ODEs in the system.

8: **v**(**ncode**) – REAL (KIND=nag_wp) array

If **ncode** > 0, **v**(i) contains the value of the component $V_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

9: **vdot**(**ncode**) – REAL (KIND=nag_wp) array

If **ncode** > 0, **vdot**(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

Note: **vdot**(i), for $i = 1, 2, \ldots,$ **ncode**, may only appear linearly as in (7) and (8).

10: **ires** – INTEGER

The form of G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$
Equation (11) must be used.

ires $= 1$
Equation (12) must be used.

Output Parameters

1: **res**(**nobc**) – REAL (KIND=nag_wp) array

res(i) must contain the ith component of G^L or G^R, depending on the value of **ibnd**, for $i = 1, 2, \ldots,$ **nobc**, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^L \dot{V}_j, \tag{11}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathbf{ncode}} H_{i,j}^L \dot{V}_j + K_i^L, \tag{12}$$

i.e., all terms in equation (7), and similarly for G_i^R.

The definitions of G^L and G^R are determined by the input value of **ires**.

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 (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 nag_pde_1d_parab_dae_keller (d03pk) returns to the calling function with the error indicator set to **ifail** $= 4$.

6: **u**(**neqn**) – REAL (KIND=nag_wp) array

The initial values of the dependent variables defined as follows:

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

u(**npts** $\times$ **npde** $+ i$) contain $V_i(t_0)$, for $i = 1, 2, \ldots,$ **ncode**.

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: **nleft** – INTEGER

The number n_a of boundary conditions at the left-hand mesh point **x**(1).

Constraint: $0 \leq$ **nleft** $\leq$ **npde**.

9: **ncode** – INTEGER

The number of coupled ODE components.

Constraint: **ncode** ≥ 0.

10: **odedef** – SUBROUTINE, supplied by the NAG Library or the user.

odedef must evaluate the functions R, which define the system of ODEs, as given in (4).

If you wish to compute the solution of a system of PDEs only (i.e., **ncode** $= 0$), **odedef** must be the string `nag_pde_1d_parab_dae_keller_remesh_fd_dummy_odedef (d03pek)`. (nag_pde_1d_parab_dae_keller_remesh_fd_dummy_odedef (d03pek) is included in the NAG Toolbox.)

```
[r, ires] = odedef(npde, t, ncode, v, vdot, nxi, xi, ucp, ucpx, ucpt, 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: **ncode** – INTEGER

The number of coupled ODEs in the system.

4: **v**(**ncode**) – REAL (KIND=nag_wp) array

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

If **ncode** > 0, **vdot**(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ **ncode**.

6: **nxi** – INTEGER

The number of ODE/PDE coupling points.

7: **xi**(**nxi**) – REAL (KIND=nag_wp) array

If **nxi** > 0, **xi**(i) contains the ODE/PDE coupling points, ξ_i, for $i = 1, 2, \ldots,$ **nxi**.

8: **ucp**(**npde**, :) – REAL (KIND=nag_wp) array

The second dimension of the array **ucp** must be at least $\max(1,$ **nxi**$)$.

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

The second dimension of the array **ucpx** must be at least $\max(1,$ **nxi**$)$.

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

The second dimension of the array **ucpt** must be at least $\max(1,$ **nxi**$)$.

If **nxi** > 0, **ucpt**(i,j) contains the value of $\dfrac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,\ldots,$ **nxi**.

11: **ires** – INTEGER

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

ires $= -1$
Equation (13) must be used.

ires $= 1$
Equation (14) must be used.

Output Parameters

1: **r**(**ncode**) – REAL (KIND=nag_wp) array

If **ncode** > 0, **r**(i) must contain the ith component of R, for $i = 1,2,\ldots,$ **ncode**, where R is defined as

$$R = -B\dot{V} - CU_t^*, \tag{13}$$

i.e., only terms depending explicitly on time derivatives, or

$$R = A - B\dot{V} - CU_t^*, \tag{14}$$

i.e., all terms in equation (4). The definition of R is determined by the input value of **ires**.

2: **ires** – INTEGER

Should usually remain unchanged. However, you may reset **ires** to force the integration function to take certain actions, as described below:

ires $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)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 nag_pde_1d_parab_dae_keller (d03pk) returns to the calling function with the error indicator set to **ifail** $= 4$.

11: **xi**(:) – REAL (KIND=nag_wp) array

The dimension of the array **xi** must be at least max$(1,$ **nxi**$)$

xi(i), for $i = 1,2,\ldots,$ **nxi**, must be set to the ODE/PDE coupling points, ξ_i.

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

12: **rtol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **rtol** must be at least 1 if **itol** $= 1$ or 2 and at least **neqn** if **itol** $= 3$ or 4

The relative local error tolerance.

Constraint: **rtol**$(i) \geq 0.0$ for all relevant i.

13: **atol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **atol** must be at least 1 if **itol** $= 1$ or 3 and at least **neqn** if **itol** $= 2$ or 4

The absolute local error tolerance.

Constraint: **atol**$(i) \geq 0.0$ for all relevant i.

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

14: **itol** – INTEGER

A value to indicate the form of the local error test. **itol** indicates to nag_pde_1d_parab_dae_keller (d03pk) 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**$(1) \times \|\mathbf{u}(i)\| +$ **atol**(1)
2	scalar	vector	**rtol**$(1) \times \|\mathbf{u}(i)\| +$ **atol**(i)
3	vector	scalar	**rtol**$(i) \times \|\mathbf{u}(i)\| +$ **atol**(1)
4	vector	vector	**rtol**$(i) \times \|\mathbf{u}(i)\| +$ **atol**(i)

In the above, e_i denotes the estimated local error for the ith component of the coupled PDE/ODE system in time, $\mathbf{u}(i)$, for $i = 1, 2, \ldots,$ **neqn**.

The choice of norm used is defined by the argument **norm_p**.

Constraint: $1 \leq$ **itol** ≤ 4.

15: **norm_p** – CHARACTER(1)

The type of norm to be used.

norm_p $=$ 'M'
Maximum norm.

norm_p $=$ 'A'
Averaged L_2 norm.

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

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

while for the maximum norm

$$\mathbf{u}_{\text{norm}} = \max_i |\mathbf{u}(i)/w_i|.$$

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

Constraint: **norm_p** $=$ 'M' or 'A'.

16: **laopt** – CHARACTER(1)

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 (i.e., **ncode** = 0).

17: **algopt**(**30**) – REAL (KIND=nag_wp) array

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

algopt(1)
Selects the ODE integration method to be used. If **algopt**(1) = 1.0, a BDF method is used and if **algopt**(1) = 2.0, a Theta method is used. The default value 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) ≤ 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 argument **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 this range then the default value is used. If the functions regard the Jacobian matrix as numerically singular then increasing **algopt**(29) towards 1.0 may help, but at the cost of increased fill-in. The default value is **algopt**(29) $= 0.1$.

algopt(30)
Used as a 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 is found to be numerically singular (see **algopt**(29)). The default value is **algopt**(30) $= 0.0001$.

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

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

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

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

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

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

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

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

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

isave(4)
Contains the order of the ODE method last used in the time integration.

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

20: **itask** – 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**.

itask $= 4$
Normal computation of output values **u** at $t =$ **tout** but without overshooting $t = t_{\mathrm{crit}}$ where t_{crit} is described under the argument **algopt**.

itask $= 5$
Take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the argument **algopt**.

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

21: **itrace** – INTEGER

The level of trace information required from nag_pde_1d_parab_dae_keller (d03pk) and the underlying ODE solver as follows:

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 $= 1$
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.

itrace $= 2$
Output from the underlying ODE solver is similar to that produced when **itrace** $= 1$, except that the advisory messages are given in greater detail.

itrace ≥ 3
Output from the underlying ODE solver is similar to that produced when **itrace** $= 2$, except that the advisory messages are given in greater detail.

You advised to set **itrace** $= 0$, unless you are experienced with Sub-chapter D02M–N.

22: **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_dae_keller (d03pk).

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

5.2 Optional Input Parameters

1: **npts** – INTEGER

Default: the dimension of the array **x**.

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

Constraint: **npts** ≥ 3.

2: **nxi** – INTEGER

Default: the dimension of the array **xi**.

The number of ODE/PDE coupling points.

Constraints:

if **ncode** $= 0$, **nxi** $= 0$;
if **ncode** > 0, **nxi** ≥ 0.

3: **neqn** – INTEGER

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

The number of ODEs in the time direction.

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

5.3 Output Parameters

1: **ts** – REAL (KIND=nag_wp)

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

2: **u**(**neqn**) – REAL (KIND=nag_wp) array

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

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 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 ODE method last used in the time integration.

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

5: **ind** – 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, (**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 [**x**(1), **x**(**npts**)],
or **npts** < 3,
or **npde** < 1,
or **nleft** not in range 0 to **npde**,
or **norm_p** $\neq$ 'A' or 'M',
or **laopt** $\neq$ 'F', 'B' or 'S',
or **itol** $\neq$ 1, 2, 3 or 4,
or **ind** $\neq$ 0 or 1,
or mesh points **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 nag_pde_1d_parab_dae_keller (d03pk),
or an element of **rtol** or **atol** < 0.0,
or corresponding elements of **atol** and **rtol** are both 0.0,
or **neqn** $\neq$ **npde** $\times$ **npts** + **ncode**.

ifail $= 2$ (*warning*)

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 $= 3$ (*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 positioning 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**, **bndary** or **odedef**, when the residual in the underlying ODE solver was being evaluated. Incorrect positioning of boundary conditions may also result in this error.

ifail $= 5$

In solving the ODE system, a singular Jacobian has been encountered. You should check their problem formulation.

ifail $= 6$ (*warning*)

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

ifail $= 7$

The values of **atol** and **rtol** are so small that the function is unable to start the integration in time.

ifail $= 8$

In either, **pdedef**, **bndary** or **odedef**, **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 **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). 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 $= 13$ (*warning*)

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 $= 14$

Not applicable.

ifail $= 15$

When using the sparse option, the value of $lisave$ or $lrsave$ was insufficient (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.

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_dae_keller (d03pk) 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 arguments, **atol** and **rtol**.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first-order by the introduction of new variables (see the example in Section 10). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite difference scheme (see nag_pde_1d_parab_fd (d03pc) or nag_pde_1d_parab_dae_fd (d03ph) for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretization scheme with upwind weighting (nag_pde_1d_parab_convdiff_dae (d03pl) for example), or the addition of a second-order artificial dissipation term.

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

9 Example

This example provides a simple coupled system of two PDEs and one ODE.

$$(V_1)^2\frac{\partial U_1}{\partial t} - xV_1\dot{V}_1U_2 - \frac{\partial U_2}{\partial x} = 0,$$

$$U_2 - \frac{\partial U_1}{\partial x} = 0,$$

$$\dot{V}_1 - V_1U_1 - U_2 - 1 - t = 0,$$

for $t \in \left[10^{-4}, 0.1 \times 2^i\right]$, for $i = 1, 2, \ldots, 5, x \in [0, 1]$. The left boundary condition at $x = 0$ is

$$U_2 = -V_1 \exp t,$$

and the right boundary condition at $x = 1$ is

$$U_2 = -V_1\dot{V}_1.$$

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

$$V_1 = t, U_1(x,t) = \exp\{t(1-x)\} - 1.0 \quad \text{and} \quad U_2(x,t) = -t\exp\{t(1-x)\},\ x \in [0,1],$$

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

This problem is exactly the same as the nag_pde_1d_parab_dae_fd (d03ph) example problem, but reduced to first-order by the introduction of a second PDE variable (as mentioned in Section 9).

9.1 Program Text

```
     function d03pk_example

fprintf('d03pk example results\n\n');

npde   = nag_int(2);
ts     = 0.0001;
tout   = 0.2;
x      = [0:0.05:1];
u(1:2:42) = exp(ts*(1-x))-1;
u(2:2:42) = -ts*exp(ts*(1-x));
u(43)  = ts;
nleft  = nag_int(1);
ncode  = nag_int(1);
xi     = [1];
rtol   = [0.0001];
atol   = [0.0001];
itol   = nag_int(1);
normt  = 'A';
laopt  = 'F';
algopt = zeros(30,1);
algopt(1)  = 1;
algopt(13) = 0.005;
rsave  = zeros(4000, 1);
isave  = zeros(100, 1, nag_int_name);
itask  = nag_int(1);
itrace = nag_int(0);
ind    = nag_int(0);
for i_t = 1:5
  tout = 2^i_t/10;

  [ts, u, rsave, isave, ind, ifail] = ...
  d03pk(...
         npde, ts, tout, @pdedef, @bndary, u, x, nleft, ncode, ...
        @odedef, xi, rtol, atol, itol, normt, laopt, algopt, ...
        rsave, isave, itask, itrace, ind);

  fprintf('\nThe solution at t = %7.4f is:\n',ts);
  fprintf('%3s%12s%9s ','x','u_1(x,t)','u_2(x,t)');
  fprintf('%5s%12s%9s ','x','u_1(x,t)','u_2(x,t)');
  fprintf('%5s%12s%9s\n','x','u_1(x,t)','u_2(x,t)');
  for i=1:7
    fprintf('%5.2f%9.4f%9.4f ',x(i),u(2*i-1),u(2*i));
    fprintf('%8.2f%9.4f%9.4f ',x(i+7),u(2*i+13),u(2*i+14));
    fprintf('%8.2f%9.4f%9.4f\n',x(i+14),u(2*i+27),u(2*i+28));
  end
end

fig1 = figure;
plot(x,u(1:2:41),x,u(2:2:42));
title({'Two PDEs coupled with ODE solution at t=3.2',...
       'using Keller Box and BDF'});
xlabel('x');
ylabel('u');
legend('u_1(x,t=3.2)','u_2(x,t=3.2)');

function [res, ires] = pdedef(npde, t, x, u, ut, ux, ncode, v, vdot, ires)
  res = zeros(npde, 1);
  if (ires == -1)
    res(1) = v(1)*v(1)*ut(1) - x*u(2)*v(1)*vdot(1);
    res(2) = 0;
  else
    res(1) = v(1)*v(1)*ut(1) - x*u(2)*v(1)*vdot(1) - ux(2);
    res(2) = u(2) - ux(1);
  end

function [res, ires] = bndary(npde, t, ibnd, nobc, u, ut, ncode, v, vdot, ires)
  res = zeros(nobc, 1);
  if (ibnd == 0)
    if (ires == -1)
```

```
      res(1) = 0;
    else
      res(1) = u(2) + v(1)*exp(t);
    end
  else
    if (ires == -1)
      res(1) = v(1)*vdot(1);
    else
      res(1) = u(2) + v(1)*vdot(1);
    end
  end

function [f, ires] = odedef(npde, t, ncode, v, vdot, nxi, xi, ucp, ucpx, ...
                            ucpt, ires)
  f = zeros(ncode, 1);
  if (ires == -1)
    f(1) = vdot(1);
  else
    f(1) = vdot(1) - v(1)*ucp(1,1) - ucp(2,1) - 1 - t;
  end
```

9.2 Program Results

```
     d03pk example results

The solution at t =  0.2000 is:
  x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)
 0.00   0.2216  -0.2443     0.35   0.1391  -0.2278     0.70   0.0621  -0.2124
 0.05   0.2095  -0.2419     0.40   0.1277  -0.2255     0.75   0.0515  -0.2103
 0.10   0.1975  -0.2395     0.45   0.1165  -0.2233     0.80   0.0410  -0.2082
 0.15   0.1855  -0.2371     0.50   0.1054  -0.2211     0.85   0.0307  -0.2061
 0.20   0.1737  -0.2347     0.55   0.0944  -0.2189     0.90   0.0204  -0.2041
 0.25   0.1621  -0.2324     0.60   0.0835  -0.2167     0.95   0.0103  -0.2020
 0.30   0.1505  -0.2301     0.65   0.0727  -0.2145     1.00   0.0002  -0.2000

The solution at t =  0.4000 is:
  x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)
 0.00   0.4920  -0.5967     0.35   0.2971  -0.5188     0.70   0.1276  -0.4510
 0.05   0.4625  -0.5849     0.40   0.2714  -0.5085     0.75   0.1053  -0.4421
 0.10   0.4335  -0.5733     0.45   0.2462  -0.4984     0.80   0.0834  -0.4333
 0.15   0.4051  -0.5620     0.50   0.2216  -0.4886     0.85   0.0620  -0.4248
 0.20   0.3773  -0.5509     0.55   0.1974  -0.4789     0.90   0.0410  -0.4163
 0.25   0.3500  -0.5400     0.60   0.1737  -0.4694     0.95   0.0203  -0.4081
 0.30   0.3233  -0.5293     0.65   0.1504  -0.4601     1.00   0.0001  -0.4000

The solution at t =  0.8000 is:
  x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)
 0.00   1.2255  -1.7804     0.35   0.6819  -1.3455     0.70   0.2711  -1.0169
 0.05   1.1382  -1.7106     0.40   0.6160  -1.2928     0.75   0.2213  -0.9770
 0.10   1.0544  -1.6435     0.45   0.5526  -1.2421     0.80   0.1734  -0.9387
 0.15   0.9738  -1.5791     0.50   0.4917  -1.1934     0.85   0.1274  -0.9019
 0.20   0.8964  -1.5171     0.55   0.4332  -1.1466     0.90   0.0832  -0.8666
 0.25   0.8220  -1.4576     0.60   0.3770  -1.1016     0.95   0.0407  -0.8326
 0.30   0.7506  -1.4005     0.65   0.3230  -1.0584     1.00  -0.0001  -0.8000

The solution at t =  1.6000 is:
  x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)
 0.00   3.9521  -7.9238     0.35   1.8279  -4.5241     0.70   0.6149  -2.5837
 0.05   3.5711  -7.3140     0.40   1.6104  -4.1761     0.75   0.4907  -2.3851
 0.10   3.2195  -6.7512     0.45   1.4096  -3.8549     0.80   0.3760  -2.2017
 0.15   2.8949  -6.2317     0.50   1.2243  -3.5584     0.85   0.2702  -2.0324
 0.20   2.5953  -5.7522     0.55   1.0532  -3.2847     0.90   0.1725  -1.8762
 0.25   2.3188  -5.3096     0.60   0.8953  -3.0321     0.95   0.0823  -1.7320
 0.30   2.0635  -4.9011     0.65   0.7495  -2.7989     1.00  -0.0010  -1.5989

The solution at t =  3.2000 is:
  x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)     x    u_1(x,t) u_2(x,t)
 0.00  23.5216 -78.4422     0.35   6.9892 -25.5334     0.70   1.6046  -8.3234
 0.05  19.8902 -66.8154     0.40   5.8070 -21.7533     0.75   1.2192  -7.0927
```

```
0.10   16.7970 -56.9137     0.45    4.7998 -18.5334     0.80    0.8908  -6.0443
0.15   14.1621 -48.4808     0.50    3.9417 -15.7905     0.85    0.6109  -5.1510
0.20   11.9176 -41.2986     0.55    3.2106 -13.4540     0.90    0.3724  -4.3900
0.25   10.0056 -35.1815     0.60    2.5877 -11.4636     0.95    0.1691  -3.7416
0.30    8.3768 -29.9713     0.65    2.0569  -9.7679     1.00   -0.0042  -3.1892
```