NAG C Library Function Document

nag_pde_parab_1d_keller_ode_remesh (d03prc)

1 Purpose

nag_pde_parab_1d_keller_ode_remesh (d03prc) 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), and automatic adaptive spatial remeshing. The spatial discretization is performed using the Keller box scheme (see Keller (1970)) 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 Specification

```
#include <nag.h>
#include <nagd03.h>

void nag_pde_parab_1d_keller_ode_remesh (Integer npde, double *ts, double tout,
     void (*pdedef)(Integer npde, double t, double x, const double u[],
          const double udot[], const double ux[], Integer ncode, const double v[],
          const double vdot[], double res[], Integer *ires, Nag_Comm *comm),
     void (*bndary)(Integer npde, double t, Integer ibnd, Integer nobc,
          const double u[], const double udot[], Integer ncode, const double v[],
          const double vdot[], double res[], Integer *ires, Nag_Comm *comm),
     void (*uvinit)(Integer npde, Integer npts, Integer nxi, const double x[],
          const double xi[], double u[], Integer ncode, double v[],
          Nag_Comm *comm),
     double u[], Integer npts, double x[], Integer nleft, 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 ucpt[], 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[], Nag_Boolean remesh, Integer nxfix, const double xfix[],
     Integer nrmesh, double dxmesh, double trmesh, Integer ipminf, double xratio,
     double con,
     void (*monitf)(double t, Integer npts, Integer npde, const double x[],
          const double u[], double fmon[], Nag_Comm *comm),
     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_keller_ode_remesh (d03prc) integrates the system of first-order PDEs and coupled ODEs given by the master equations:

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

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

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

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

where $P_{i,j}$, $Q_{i,j}$ and R_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 F_i may only depend linearly on time derivatives. Hence equation (2) may be written more precisely as

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

where $F = \left[F_1,\ldots,F_{\mathbf{ncode}}\right]^{\mathrm{T}}$, 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 and 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 the user-supplied function **odedef**.)

The integration in time is from t_0 to t_{out}, over the space interval $a \le x \le b$, where $a = x_1$ and $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$ defined initially by you and (possibly) adapted automatically during the integration according to user-specified criteria.

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

The initial $(t = t_0)$ values of the functions $U(x,t)$ and $V(t)$ must be specified in a function **uvinit** supplied by you. Note that **uvinit** will be called again following any remeshing, and so $U(x,t_0)$ should be specified for **all** values of x in the interval $a \le x \le b$, and not just the initial mesh points.

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 master equation 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 functions. 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:

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

at the left-hand boundary, and

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

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

The boundary conditions must be specified in a function **bndary** provided by you.

The problem is subject to the following restrictions:

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

(ii) $t_0 < t_{\text{out}}$, 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 $\mathbf{x}[i-1]$, for $i = 1, 2, \ldots,$ **npts**, by calling the user-supplied function **pdedef** for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the fixed mesh points specified by **xfix**;

(iv) 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.

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

The first-order 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 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.

The adaptive space remeshing can be used to generate meshes that automatically follow the changing time-dependent nature of the solution, generally resulting in a more efficient and accurate solution using fewer mesh points than may be necessary with a fixed uniform or non-uniform mesh. Problems with travelling wavefronts or variable-width boundary layers for example will benefit from using a moving adaptive mesh. The discrete time-step method used here (developed by Furzeland (1984)) automatically creates a new mesh based on the current solution profile at certain time-steps, and the solution is then interpolated onto the new mesh and the integration continues.

The method requires you to supply a function **monitf** which specifies in an analytic or numeric form the particular aspect of the solution behaviour you wish to track. This so-called monitor function is used to choose a mesh which equally distributes the integral of the monitor function over the domain. A typical choice of monitor function is the second space derivative of the solution value at each point (or some combination of the second space derivatives if more than one solution component), which results in refinement in regions where the solution gradient is changing most rapidly.

You must specify the frequency of mesh updates along with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and you are encouraged to experiment with the different options in order to achieve an efficient solution which adequately tracks the desired features of the solution.

Note that unless the monitor function for the initial solution values is zero at all user-specified initial mesh points, a new initial mesh is calculated and adopted according to the user-specified remeshing criteria. The function **uvinit** will then be called again to determine the initial solution values at the new mesh points (there is no interpolation at this stage) and the integration proceeds.

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

Furzeland R M (1984) The construction of adaptive space meshes *TNER.85.022* Thornton Research Centre, Chester

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 Arguments

1: **npde** – Integer *Input*

On entry: the number of PDEs to be solved.

Constraint: **npde** ≥ 1.

2: **ts** – double * *Input/Output*

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

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

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

3: **tout** – double *Input*

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

4: **pdedef** – function, supplied by the user *External Function*

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_parab_1d_keller_ode_remesh (d03prc).

Its specification is:

```
void pdedef (Integer npde, double t, double x, const double u[],
     const double udot[], const double ux[], Integer ncode, const double v[],
     const double vdot[], double res[], 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: **udot**[**npde**] – const double *Input*

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

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

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

7: **ncode** – Integer *Input*

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

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

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

9: **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**.

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

On exit: **res**$[i-1]$ 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

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

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

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

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

On entry: the form of G_i that must be returned in the array **res**. If **ires** $= -1$, then equation (9) above must be used. If **ires** $= 1$, then equation (10) above must be used.

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

ires $= 2$

Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to **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. 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_parab_1d_keller_ode_remesh (d03prc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

12: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **pdedef**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) these pointers may be allocated memory by the user and initialized with various quantities for use by **pdedef** when called from nag_pde_parab_1d_keller_ode_remesh (d03prc).

5: **bndary** – function, supplied by the user *External Function*

bndary must evaluate the functions G_i^L and G_i^R which describe the boundary conditions, as given in (5) and (6).

Its specification is:

```
void bndary (Integer npde, double t, Integer ibnd, Integer nobc,
      const double u[], const double udot[], Integer ncode, const double v[],
      const double vdot[], double res[], 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: **ibnd** – Integer *Input*

On entry: 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 of the right-hand boundary condition at $x = b$.

4: **nobc** – Integer *Input*

On entry: specifies the number n_a of boundary conditions at the boundary specified by **ibnd**.

5: **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 $i = 1, 2, \ldots,$ **npde**.

6: **udot**[**npde**] – const double *Input*

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

7: **ncode** – Integer *Input*

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

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

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

9: **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: **vdot**$[i-1]$, for $i = 1, 2, \ldots,$ **ncode**, may only appear linearly as in (11) and (12).

10: **res**[**nobc**] – double *Output*

On exit: **res**$[i-1]$ 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

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

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**.

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

On entry: the form of G_i^L (or G_i^R) that must be returned in the array **res**. If **ires** $= -1$, then equation (11) above must be used. If **ires** $= 1$, then equation (12) above must be used.

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

ires $= 2$

Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to **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. 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_parab_1d_keller_ode_remesh (d03prc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

12: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **bndary**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) these pointers may be allocated memory by the user and initialized with various quantities for use by **bndary** when called from nag_pde_parab_1d_keller_ode_remesh (d03prc).

6: **uvinit** – function, supplied by the user *External Function*

uvinit must supply the initial $(t = t_0)$ values of $U(x, t)$ and $V(t)$ for all values of x in the interval $[a, b]$.

Its specification is:

```
void uvinit (Integer npde, Integer npts, Integer nxi, const double x[],
     const double xi[], double u[], Integer ncode, double v[], Nag_Comm *comm)
```

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **npts** – Integer *Input*

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

3: **nxi** – Integer *Input*

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

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

On entry: the current mesh. **x**$[i-1]$ contains the value of x_i, for $i = 1,2,\ldots,$ **npts**.

5: **xi**[**nxi**] – const double *Input*

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

6: **u**[**npde** $\times$ **npts**] – double *Output*

On exit: **u**[**npde** $\times j + i$] contains the value of the component $U_i(x_j,t_0)$, for $i = 1,2,\ldots,$ **npde**; $j = 1,2,\ldots,$ **npts**.

7: **ncode** – Integer *Input*

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

8: **v**[**ncode**] – double *Output*

On exit: **v**$[i-1]$ must contain the value of component $V_i(t_0)$, for $i = 1,2,\ldots,$ **ncode**.

9: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **uvinit**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) these pointers may be allocated memory by the user and initialized with various quantities for use by **uvinit** when called from nag_pde_parab_1d_keller_ode_remesh (d03prc).

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

On entry: if **ind** $= 1$, the value of **u** must be unchanged from the previous call.

On exit: **u**[**npde** $\times (j-1) + i - 1$] contains the computed solution $U_i(x_j,t)$, for $i = 1,2,\ldots,$ **npde**; $j = 1,2,\ldots,$ **npts**, and **u**[**npts** $\times$ **npde** $+ k - 1$] contains $V_k(t)$, 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**] – double *Input/Output*

On entry: the initial 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: $\mathbf{x}[0] < \mathbf{x}[1] < \cdots < \mathbf{x}[\mathbf{npts} - 1]$.

On exit: the final values of the mesh points.

10: **nleft** – Integer *Input*

On entry: the number n_a of boundary conditions at the left-hand mesh point **x**[0].

Constraint: $0 \leq \mathbf{nleft} \leq \mathbf{npde}$.

11: **ncode** – Integer *Input*

On entry: the number of coupled ODE components.

Constraint: $\mathbf{ncode} \geq 0$.

12: **odedef** – function, supplied by the user *External Function*

odedef must evaluate the functions F, which define the system of ODEs, as given in (4). If you wish to compute the solution of a system of PDEs only (i.e., **ncode** = 0), **odedef** must be the dummy function d03pek. (d03pek is included in the NAG C Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
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 ucpt[], 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 point, ξ_i, for $i = 1, 2, \ldots,$ **nxi**.

8: **ucp**[**npde** × **nxi**] – const double *Input*

On entry: **ucp**[**npde** × $j + i$] 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** × **nxi**] – const double *Input*

On entry: **ucpx**[**npde** × $j + i$] 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**; $j = 1, 2, \ldots,$ **nxi**.

10: **ucpt**[**npde** × **nxi**] – const double *Input*

On entry: **ucpt**[**npde** × $j + i$] contains the value of $\dfrac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ **npde**; $j = 1, 2, \ldots,$ **nxi**.

11: **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

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

that is, only terms depending explicitly on time derivatives, or

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

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

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

On entry: the form of **f** that must be returned in the array **f**. If **ires** $= -1$, then equation (13) above must be used. If **ires** $= 1$, then equation (14) above must be used.

On exit: 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 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. 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_parab_1d_keller_ode_remesh (d03prc) returns to the calling function with the error indicator set to **fail.code** = **NE_FAILED_DERIV**.

13: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **odedef**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) these pointers may be allocated memory by the user and initialized with various quantities for use by **odedef** when called from nag_pde_parab_1d_keller_ode_remesh (d03prc).

13: **nxi** – Integer *Input*

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

Constraints:

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

14: **xi**[dim] – const double *Input*

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

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

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

15: **neqn** – Integer *Input*

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

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

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

Note: the dimension, *dim*, of the array **rtol** must be at least

1 when **itol** $= 1$ or 2;
neqn when **itol** $= 3$ or 4.

On entry: the relative local error tolerance.

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

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

Note: the dimension, *dim*, of the array **atol** must be at least

1 when **itol** $= 1$ or 3;
neqn when **itol** $= 2$ or 4.

On entry: the absolute local error tolerance.

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

18: **itol** – Integer *Input*

A value to indicate the form of the local error test. **itol** indicates to nag_pde_parab_1d_keller_ode_remesh (d03prc) 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:

On entry:

itol	**rtol**	**atol**	w_i
1	scalar	scalar	**rtol**$[0] \times \lvert\textbf{u}[i-1]\rvert +$ **atol**$[0]$
2	scalar	vector	**rtol**$[0] \times \lvert\textbf{u}[i-1]\rvert +$ **atol**$[i-1]$
3	vector	scalar	**rtol**$[i-1] \times \lvert\textbf{u}[i-1]\rvert +$ **atol**$[0]$
4	vector	vector	**rtol**$[i-1] \times \lvert\textbf{u}[i-1]\rvert +$ **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 argument **norm**, see below.

Constraint: $1 \leq$ **itol** ≤ 4.

19: **norm** – Nag_NormType *Input*

On entry: the type of norm to be used.

norm = **Nag_MaxNorm**

Maximum norm.

norm = **Nag_TwoNorm**

Averaged L_2 norm.

If U_{norm} denotes the norm of the vector **u** of length **neqn**, then for the averaged L_2 norm

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

while for the maximum norm

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

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

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

20: **laopt** – Nag_LinAlgOption *Input*

On entry: the type of matrix algebra required.

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

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

On entry: may be set to control various options available in the integrator. If you wish to employ all the default options, then **algopt**[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 argument **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 initialize 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, i.e., **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]

Used as a relative pivot threshold during subsequent Jacobian decompositions (see **algopt**[28]) below which an internal error is invoked. **algopt**[29] must be greater than zero, otherwise the default value is used. 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.

22: **remesh** – Nag_Boolean *Input*

On entry: indicates whether or not spatial remeshing should be performed.

remesh = **Nag_True**

Indicates that spatial remeshing should be performed as specified.

remesh = **Nag_False**

Indicates that spatial remeshing should be suppressed.

Note: **remesh** should **not** be changed between consecutive calls to nag_pde_parab_1d_keller_ode_remesh (d03prc). Remeshing can be switched off or on at specified times by using appropriate values for the arguments **nrmesh** and **trmesh** at each call.

23: **nxfix** – Integer *Input*

On entry: the number of fixed mesh points.

Constraint: $0 \leq \mathbf{nxfix} \leq \mathbf{npts} - 2$

Note: the end points **x**[0] and **x**[**npts** − 1] are fixed automatically and hence should not be specified as fixed points.

24: **xfix**[dim] – const double *Input*

Note: the dimension, *dim*, of the array **xfix** must be at least $\max(1, \mathbf{nxfix})$.

On entry: **xfix**[$i-1$], for $i = 1, 2, \ldots, \mathbf{nxfix}$, must contain the value of the x co-ordinate at the ith fixed mesh point.

Constraint: $\mathbf{xfix}[i-1] < \mathbf{xfix}[i]$, for $i = 1, 2, \ldots, \mathbf{nxfix} - 1$, and each fixed mesh point must coincide with a user-supplied initial mesh point, that is $\mathbf{xfix}[i-1] = \mathbf{x}[j-1]$ for some j, $2 \leq j \leq \mathbf{npts} - 1$.

Note: the positions of the fixed mesh points in the array **x** remain fixed during remeshing, and so the number of mesh points between adjacent fixed points (or between fixed points and end points) does not change. You should take this into account when choosing the initial mesh distribution.

25: **nrmesh** – Integer *Input*

On entry: indicates the form of meshing to be performed.

nrmesh < 0

Indicates that a new mesh is adopted according to the argument **dxmesh** below. The mesh is tested every |**nrmesh**| timesteps.

nrmesh $= 0$

Indicates that remeshing should take place just once at the end of the first time step reached when $t >$ **trmesh** (see below).

nrmesh > 0

Indicates that remeshing will take place every **nrmesh** time steps, with no testing using **dxmesh**.

Note: **nrmesh** may be changed between consecutive calls to nag_pde_parab_1d_keller_ode_remesh (d03prc) to give greater flexibility over the times of remeshing.

26: **dxmesh** – double *Input*

On entry: determines whether a new mesh is adopted when **nrmesh** is set less than zero. A possible new mesh is calculated at the end of every $|\mathbf{nrmesh}|$ time steps, but is adopted only if

$$x_i^{\text{new}} > x_i^{\text{old}} + \mathbf{dxmesh} \times \left(x_{i+1}^{\text{old}} - x_i^{\text{old}}\right),$$

or

$$x_i^{\text{new}} < x_i^{\text{old}} - \mathbf{dxmesh} \times \left(x_i^{\text{old}} - x_{i-1}^{\text{old}}\right).$$

dxmesh thus imposes a lower limit on the difference between one mesh and the next.

Constraint: $\mathbf{dxmesh} \geq 0.0$.

27: **trmesh** – double *Input*

On entry: specifies when remeshing will take place when **nrmesh** is set to zero. Remeshing will occur just once at the end of the first time step reached when t is greater than **trmesh**.

Note: **trmesh** may be changed between consecutive calls to nag_pde_parab_1d_keller_ode_remesh (d03prc) to force remeshing at several specified times.

28: **ipminf** – Integer *Input*

On entry: the level of trace information regarding the adaptive remeshing.

ipminf $= 0$

No trace information.

ipminf $= 1$

Brief summary of mesh characteristics.

ipminf $= 2$

More detailed information, including old and new mesh points, mesh sizes and monitor function values.

Constraint: $0 \leq \mathbf{ipminf} \leq 2$.

29: **xratio** – double *Input*

On entry: input bound on adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The remeshing functions will attempt to ensure that

$$(x_i - x_{i-1})/\mathbf{xratio} < x_{i+1} - x_i < \mathbf{xratio} \times (x_i - x_{i-1}).$$

Suggested value: $\mathbf{xratio} = 1.5$.

Constraint: $\mathbf{xratio} > 1.0$.

30: **con** – double *Input*

On entry: an input bound on the sub-integral of the monitor function $F^{\text{mon}}(x)$ over each space step. The remeshing functions will attempt to ensure that

$$\int_{x_1}^{x_{i+1}} F^{\text{mon}}(x)dx \leq \mathbf{con} \int_{x_1}^{x_{\mathbf{npts}}} F^{\text{mon}}(x)dx,$$

(see Furzeland (1984)). **con** gives you more control over the mesh distribution e.g., decreasing **con** allows more clustering. A typical value is $2/(\mathbf{npts} - 1)$, but you are encouraged to experiment with different values. Its value is not critical and the mesh should be qualitatively correct for all values in the range given below.

Suggested value: **con** $= 2.0/(\mathbf{NPTS} - 1)$.

Constraint: $0.1/(\mathbf{npts} - 1) \leq \mathbf{con} \leq 10.0/(\mathbf{npts} - 1)$.

31: **monitf** – function, supplied by the user *External Function*

monitf must supply and evaluate a remesh monitor function to indicate the solution behaviour of interest.

If you specify **remesh** = **Nag_False**, i.e., no remeshing, then **monitf** will not be called and the dummy function d03pel may be used for **monitf**. (d03pel is included in the NAG C Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
void monitf (double t, Integer npts, Integer npde, const double x[],
     const double u[], double fmon[], Nag_Comm *comm)
```

1: **t** – double *Input*

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

2: **npts** – Integer *Input*

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

3: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

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

On entry: the current mesh. $\mathbf{x}[i-1]$ contains the value of x_i, for $i = 1, 2, \ldots, \mathbf{npts}$.

5: **u**[**npde** × **npts**] – const double *Input*

On entry: $\mathbf{u}[\mathbf{npde} \times j + i]$ contains the value of $U_i(x, t)$ at $x = \mathbf{x}[j-1]$ and time t, for $i = 1, 2, \ldots, \mathbf{npde}$; $j = 1, 2, \ldots, \mathbf{npts}$.

6: **fmon**[**npts**] – double *Output*

On exit: $\mathbf{fmon}[i-1]$ must contain the value of the monitor function $F^{\mathrm{mon}}(x)$ at mesh point $x = \mathbf{x}[i-1]$.

7: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **monitf**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) these pointers may be allocated memory by the user and initialized with various quantities for use by **monitf** when called from nag_pde_parab_1d_keller_ode_remesh (d03prc).

32: **rsave**[**lrsave**] – double *Communication Array*

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

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

33: **lrsave** – Integer *Input*

On entry: the dimension of the array **rsave** as declared in the function from which nag_pde_parab_1d_keller_ode_remesh (d03prc) 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

ml and mu are the lower and upper half bandwidths given by **npde** + **nleft** $- 1$, and
$mu = 2 \times$ **npde** $-$ **nleft** $- 1$, for problems involving PDEs only, and
$ml = mu =$ **neqn** $- 1$, for coupled PDE/ODE problems.

$nwkres =$ **npde** $\times (3 \times$ **npde** $+ 6 \times$ **nxi** + **npts** $+ 15) +$ **nxi** + **ncode** $+ 7 \times$ **npts** $+$ **nxfix** $+ 1$, when **ncode** > 0 and **nxi** > 0, and
$nwkres =$ **npde** $\times (3 \times$ **npde** + **npts** $+ 21) +$ **ncode** $+ 7 \times$ **npts** + **nxfix** $+ 2$, when **ncode** > 0 and **nxi** $= 0$, and
$nwkres =$ **npde** $\times (3 \times$ **npde** + **npts** $+ 21) + 7 \times$ **npts** + **nxfix** $+ 3$, when **ncode** $= 0$.

$lenode = (6 +$ 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 using the sparse option, 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**.

34: **isave**[**lisave**] – Integer *Communication 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[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 evaluating 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 ODE method last used in the time integration.

isave[4]

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.

The rest of the array is used as workspace.

35: **lisave** – Integer *Input*

On entry: the dimension of the array **isave** as declared in the function from which nag_pde_parab_1d_keller_ode_remesh (d03prc) is called. Its size depends on the type of matrix algebra selected:

if **laopt** = **Nag_LinAlgFull**, **lisave** $\geq 25 +$ **nxfix**;
if **laopt** = **Nag_LinAlgBand**, **lisave** $\geq$ **neqn** $+ 25 +$ **nxfix**;
if **laopt** = **Nag_LinAlgSparse**, **lisave** $\geq 25 \times$ **neqn** $+ 25 +$ **nxfix**.

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**.

36: **itask** – Integer *Input*

On entry: 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: $1 \leq$ **itask** ≤ 5.

37: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_pde_parab_1d_keller_ode_remesh (d03prc) and the underlying ODE solver as follows:

itrace ≤ -1

No output is generated.

itrace $= 0$

Only warning messages from the PDE solver are printed .

itrace $= 1$

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.

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

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

38: **outfile** – const 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.

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

On entry: 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 arguments **tout** and **fail** and the remeshing arguments **nrmesh**, **dxmesh**, **trmesh**, **xratio** and **con** may be reset between calls to nag_pde_parab_1d_keller_ode_remesh (d03prc).

Constraint: $0 \leq$ **ind** ≤ 1.

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

40: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 2.2.1.1 of the Essential Introduction).

41: **saved** – Nag_D03_Save * *Communication Structure*

Note: **saved** is a NAG defined type (see Section 2.2.1.1 of the Essential Introduction).

saved must remain unchanged following a previous call to a d03 function and prior to any subsequent call to a d03 function.

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

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ACC_IN_DOUBT

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

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_FAILED_DERIV

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

NE_FAILED_START

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

NE_FAILED_STEP

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

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

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

NE_INCOMPAT_PARAM

On entry, **con** $< 0.1/($**npts** $- 1)$: **con** $= \langle value \rangle$, **npts** $= \langle value \rangle$.

On entry, **con** $> 10.0/($**npts** $- 1)$: **con** $= \langle value \rangle$, **npts** $= \langle value \rangle$.

On entry, the point **xfix**$[i-1]$ does not coincide with any **x**$[j-1]$: $i = \langle value \rangle$, **xfix**$[i-1] = \langle value \rangle$.

NE_INT

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

On entry, **ipminf** is not equal to 0, 1, or 2: **ipminf** $= \langle value \rangle$.

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

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

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

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

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

On entry, **npde** $= \langle value \rangle$.
Constraint: **npde** ≥ 1.

On entry, **npts** $= \langle value \rangle$.
Constraint: **npts** ≥ 3.

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

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

NE_INT_2

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

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

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

On entry, **nleft** $>$ **npde**: **nleft** $= \langle value \rangle$, **npde** $= \langle value \rangle$.

On entry, **nxfix** $>$ **npts** $- 2$: **nxfix** $= \langle value \rangle$, **npts** $= \langle value \rangle$.

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

NE_INT_4

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_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_CLOSE_FILE

Cannot close file $\langle value \rangle$.

NE_NOT_STRICTLY_INCREASING

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

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

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

NE_NOT_WRITE_FILE

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

NE_REAL

On entry, **dxmesh** $= \langle value \rangle$.
Constraint: **dxmesh** ≥ 0.0.

On entry, **xratio** $= \langle value \rangle$.
Constraint: **xratio** > 1.0.

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, **tout** − **ts** is too small: **tout** $= \langle value \rangle$, **ts** $= \langle value \rangle$.

On entry, **tout** $\leq$ **ts**: **tout** $= \langle value \rangle$, **ts** $= \langle value \rangle$.

NE_REAL_ARRAY

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

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

NE_REMESH_CHANGED

remesh has been changed between calls to nag_pde_parab_1d_fd_ode_remesh (d03ppc).

NE_SING_JAC

Singular Jacobian of ODE system. Check problem formulation.

NE_USER_STOP

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

NE_ZERO_WTS

Zero error weights encountered during time integration.

7 Accuracy

nag_pde_parab_1d_keller_ode_remesh (d03prc) 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 9). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference

scheme (nag_pde_parab_1d_fd_ode_remesh (d03ppc) 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_parab_1d_cd_ode_remesh (d03psc) for example), or the addition of a second-order artificial dissipation term.

The time taken depends on the complexity of the system, the accuracy requested, and the frequency of the mesh updates. For a given system with fixed accuracy and mesh-update frequency it is approximately proportional to **neqn**.

9 Example

This example is the first-order system

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

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

The initial conditions are

$$\begin{aligned} U_1(x,0) &= e^x, \\ U_2(x,0) &= x^2 + \sin\left(2\pi x^2\right), \end{aligned}$$

and the Dirichlet boundary conditions for U_1 at $x = 0$ and U_2 at $x = 1$ are given by the exact solution:

$$\begin{aligned} U_1(x,t) &= \tfrac{1}{2}\left\{e^{x+t} + e^{x-3t}\right\} + \tfrac{1}{4}\left\{\sin\left(2\pi(x-3t)^2\right) - \sin\left(2\pi(x+t)^2\right)\right\} + 2t^2 - 2xt, \\ U_2(x,t) &= e^{x-3t} - e^{x+t} + \tfrac{1}{2}\left\{\sin\left(2\pi(x-3t)^2\right) + \sin\left(2\pi(x+t)^2\right)\right\} + x^2 + 5t^2 - 2xt. \end{aligned}$$

9.1 Program Text

```
/* nag_pde_parab_1d_keller_ode_remesh (d03prc) 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>
#include <nagx01.h>
static void pdedef(Integer, double, double, const double[], const double[],
                           const double[], Integer, const double[],
                           const double[], double[], Integer *, Nag_Comm *);
static void bndary(Integer, double, Integer, Integer, const double[],
                           const double[], Integer, const double[],
                           const double[], double[], Integer *, Nag_Comm *);
static void uvinit(Integer, Integer, Integer, const double[],
                           const double[], double[], Integer, double[],
                           Nag_Comm *);
static void monitf(double, Integer, Integer, const double[],
                           const double[], double[], Nag_Comm *);
static void exact(double, Integer, Integer, double *, double *);

#define UE(I,J) ue[npde*((J)-1)+(I)-1]
#define U(I,J) u[npde*((J)-1)+(I)-1]
```

```
#define UOUT(I,J,K) uout[npde*(intpts*((K)-1)+(J)-1)+(I)-1]


int main(void)
{
  const Integer npde=2, npts=61, ncode=0, nxi=0, nxfix=0, nleft=1,
    itype=1, intpts=5, neqn=npde*npts+ncode, lisave=25+nxfix,
    nwkres=npde*(npts+3*npde+21)+7*npts+nxfix+3, lenode=11*neqn+50,
    lrsave=neqn*neqn+neqn+nwkres+lenode;
  double con, dxmesh, tout, trmesh, ts, xratio;
  Integer exit_status, i, ind, ipminf, it, itask, itol, itrace, nrmesh;
  Nag_Boolean remesh, theta;
  double *algopt=0, *atol=0, *rsave=0, *rtol=0, *u=0, *ue=0,
    *uout=0, *x=0, *xfix=0, *xi=0, *xout=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)) ||
       !(rsave = NAG_ALLOC(lrsave, double)) ||
       !(rtol = NAG_ALLOC(1, double)) ||
       !(u = NAG_ALLOC(npde*npts, double)) ||
       !(ue = NAG_ALLOC(npde*npts, double)) ||
       !(uout = NAG_ALLOC(npde*intpts*itype, double)) ||
       !(x = NAG_ALLOC(npts, double)) ||
       !(xfix = NAG_ALLOC(1, double)) ||
       !(xi = NAG_ALLOC(1, double)) ||
       !(xout = NAG_ALLOC(intpts, double)) ||
       !(isave = NAG_ALLOC(lisave, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = 1;
      goto END;
    }

  INIT_FAIL(fail);
  exit_status = 0;

  Vprintf("nag_pde_parab_1d_keller_ode_remesh (d03prc) Example Program"
          " Results\n\n");
  itrace = 0;
  itol = 1;
  atol[0] = 5.0e-5;
  rtol[0] = atol[0];

  Vprintf("  Accuracy requirement =%10.3e", atol[0]);
  Vprintf(" Number of points = %3ld\n\n", npts);

  /* Set remesh parameters */

  remesh = Nag_TRUE;
  nrmesh = 3;
  dxmesh = 0.0;
  trmesh = 0.0;
  con = 5.0/(npts-1.0);
  xratio = 1.2;
  ipminf = 0;
  Vprintf(" Remeshing every %3ld time steps\n\n", nrmesh);

  /* Initialise mesh */

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

  xout[0] = 0.0;
  xout[1] = 0.25;
  xout[2] = 0.5;
  xout[3] = 0.75;
```

```
  xout[4] = 1.0;
  Vprintf(" x        ");

  for (i = 0; i < intpts; ++i)
    {
      Vprintf("%10.4f", xout[i]);
      Vprintf((i+1)%5 == 0 || i == 4 ?"\n":" ");
    }
  Vprintf("\n\n");

  xi[0] = 0.0;
  ind = 0;
  itask = 1;

  /* Set theta to TRUE if the Theta integrator is required */

  theta = Nag_FALSE;
  for (i = 0; i < 30; ++i) algopt[i] = 0.0;
  if (theta)
    {
      algopt[0] = 2.0;
      algopt[5] = 2.0;
      algopt[6] = 1.0;
    }

  /* Loop over output value of t */

  ts = 0.0;
  tout = 0.0;

  for (it = 0; it < 5; ++it)
    {
      tout = 0.05*(it+1);

      /* nag_pde_parab_1d_keller_ode_remesh (d03prc).
       * General system of first-order PDEs, coupled DAEs, method
       * of lines, Keller box discretisation, remeshing, one space
       * variable
       */
      nag_pde_parab_1d_keller_ode_remesh(npde, &ts, tout, pdedef, bndary,
                                         uvinit, u, npts, x, nleft, ncode,
                                         d03pek, nxi, xi, neqn, rtol, atol,
                                         itol, Nag_TwoNorm, Nag_LinAlgFull,
                                         algopt, remesh, nxfix, xfix, nrmesh,
                                         dxmesh, trmesh, ipminf, xratio, con,
                                         monitf, rsave, lrsave, isave, lisave,
                                         itask, itrace, 0, &ind, &comm, &saved,
                                         &fail);

      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_pde_parab_1d_keller_ode_remesh"
                  " (d03prc).\n%s\n", fail.message);
          exit_status = 1;
          goto END;
        }

      /* Interpolate at output points */

      /* nag_pde_interp_1d_fd (d03pzc).
       * PDEs, spatial interpolation with nag_pde_parab_1d_fd
       * (d03pcc), nag_pde_parab_1d_keller (d03pec),
       * nag_pde_parab_1d_cd (d03pfc), nag_pde_parab_1d_fd_ode
       * (d03phc), nag_pde_parab_1d_keller_ode (d03pkc),
       * nag_pde_parab_1d_cd_ode (d03plc),
       * nag_pde_parab_1d_fd_ode_remesh (d03ppc),
       * nag_pde_parab_1d_keller_ode_remesh (d03prc) or
       * nag_pde_parab_1d_cd_ode_remesh (d03psc)
       */
      nag_pde_interp_1d_fd(npde, 0, u, npts, x, xout, intpts, itype, uout,
                           &fail);
```

```
      if (fail.code != NE_NOERROR)
        {
          Vprintf("Error from nag_pde_interp_1d_fd (d03pzc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      /* Check against exact solution */

      exact(ts, npde, intpts, xout, ue);

      Vprintf(" t = %6.3f\n", ts);
      Vprintf(" Approx u1");

      for (i = 1; i <= intpts; ++i)
        {
          Vprintf("%10.4f", UOUT(1,i,1));
          Vprintf(i%5 == 0 || i == 5 ?"\n":"");
        }

      Vprintf(" Exact  u1");

      for (i = 1; i <= 5; ++i)
        {
          Vprintf("%10.4f", UE(1,i));
          Vprintf(i%5 == 0 || i == 5 ?"\n":"");
        }

      Vprintf(" Approx u2");

      for (i = 1; i <= 5; ++i)
        {
          Vprintf("%10.4f", UOUT(2,i,1));
          Vprintf(i%5 == 0 || i == 5 ?"\n":"");
        }

      Vprintf(" Exact  u2");

      for (i = 1; i <= 5; ++i)
        {
          Vprintf("%10.4f", UE(2,i));
          Vprintf(i%5 == 0 || i == 5 ?"\n":"");
        }

      Vprintf("\n");
    }

  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 (rsave) NAG_FREE(rsave);
  if (rtol) NAG_FREE(rtol);
  if (u) NAG_FREE(u);
  if (ue) NAG_FREE(ue);
  if (uout) NAG_FREE(uout);
  if (x) NAG_FREE(x);
  if (xfix) NAG_FREE(xfix);
  if (xi) NAG_FREE(xi);
  if (xout) NAG_FREE(xout);
  if (isave) NAG_FREE(isave);

  return exit_status;
}
static void uvinit(Integer npde, Integer npts, Integer nxi, const double x[],
```

```
                           const double xi[], double u[], Integer ncode,
                           double v[], Nag_Comm *comm)
{
  Integer i;

  for (i = 1; i <= npts; ++i)
    {
      U(1, i) = exp(x[i-1]);
      U(2, i) = x[i-1]*x[i-1] + sin(2.0*nag_pi*(x[i-1]*x[i-1]));
    }
  return;
}
static void pdedef(Integer npde, double t, double x, const double u[],
                           const double udot[], const double ux[],
                           Integer ncode, const double v[],
                           const double vdot[], double res[], Integer *ires,
                           Nag_Comm *comm)
{
  if (*ires == -1)
    {
      res[0] = udot[0];
      res[1] = udot[1];
    } else {
      res[0] = udot[0] + ux[0] + ux[1];
      res[1] = udot[1] + 4.0*ux[0] + ux[1];
    }
  return;
}
static void bndary(Integer npde, double t, Integer ibnd, Integer nobc,
                           const double u[], const double udot[],
                           Integer ncode, const double v[],
                           const double vdot[], double res[], Integer *ires,
                           Nag_Comm *comm)
{
  double pp;

  pp = 2.0*nag_pi;

  if (ibnd == 0)
    {
      if (*ires == -1)
        {
          res[0] = 0.0;
        } else {
          res[0] = u[0] - 0.5*(exp(t) + exp(-3.0*t))
            -0.25*(sin(9.0*pp*t*t) - sin(pp*t*t)) - 2.0*t*t;
        }
    } else {
      if (*ires == -1) {
        res[0] = 0.0;
      } else {
        res[0] = u[1] - (exp(1.0-3.0*t) - exp(t + 1.0)
                         + 0.5*(sin(pp*(1.0-3.0*t)*(1.0-3.0*t))
                                + sin(pp*(t+1.0)*(t+1.0)))
                         + 1.0 + 5.0*t*t - 2.0*t);
      }
    }
  return;
}
static void monitf(double t, Integer npts, Integer npde, const double x[],
                           const double u[], double fmon[], Nag_Comm *comm)
{
  double d2x1, d2x2, h1, h2, h3;
  Integer i;

  for (i = 2; i <= npts-1; ++i)
    {
      h1 = x[i - 1] - x[i - 2];
      h2 = x[i] - x[i - 1];
      h3 = 0.5*(x[i] - x[i - 2]);
```

```
      /* Second derivatives */

      d2x1 = fabs(((U(1,i+1)-U(1,i))/h2-(U(1,i)-U(1,i-1))/h1)/h3);
      d2x2 = fabs(((U(2,i+1)-U(2,i))/h2-(U(2,i)-U(2,i-1))/h1)/h3);
      fmon[i-1] = d2x1; if (d2x2 > d2x1) fmon[i-1] = d2x2;
    }
  fmon[0] = fmon[1];
  fmon[npts-1] = fmon[npts-2];

  return;
}
static void exact(double t, Integer npde, Integer npts, double *x,
                  double *u)
{
  /* Exact solution (for comparison purposes) */

  double pp;
  Integer i;

  pp = 2.0*nag_pi;
  for (i = 1; i <= npts; ++i)
    {
      U(1, i) = 0.5*(exp(x[i-1]+t) + exp(x[i-1]-3.0*t))
        + 0.25*(sin(pp*(x[i-1]-3.0*t)*(x[i-1]-3.0*t))
                - sin(pp*(x[i-1]+t)*(x[i-1]+t)))
        + 2.0*t*t - 2.0*x[i-1]*t;

      U(2, i) = exp(x[i-1]-3.0*t) - exp(x[i-1]+t) +
        0.5*(sin(pp*((x[i-1]-3.0*t)*(x[i-1]-3.0*t)))
             + sin(pp*((x[i-1]+t)*(x[i-1]+t))))
        + x[i-1]*x[i-1] +  5.0*t*t - 2.0*x[i-1]*t;
    }
  return;
}
```

9.2 Program Data

None.

9.3 Program Results

```
nag_pde_parab_1d_keller_ode_remesh (d03prc) Example Program Results

  Accuracy requirement = 5.000e-05 Number of points =  61

 Remeshing every   3 time steps

 x            0.0000     0.2500     0.5000     0.7500     1.0000


 t =  0.050
 Approx u1    0.9923     1.0894     1.4686     2.3388     2.1071
 Exact  u1    0.9923     1.0893     1.4686     2.3391     2.1073
 Approx u2   -0.0997     0.1057     0.7180     0.0967     0.2021
 Exact  u2   -0.0998     0.1046     0.7193     0.0966     0.2022

 t =  0.100
 Approx u1    1.0613     0.9856     1.3120     2.3084     2.1039
 Exact  u1    1.0613     0.9851     1.3113     2.3092     2.1025
 Approx u2   -0.0150    -0.0481     0.1075    -0.3240     0.3753
 Exact  u2   -0.0150    -0.0495     0.1089    -0.3235     0.3753

 t =  0.150
 Approx u1    1.1485     0.9763     1.2658     2.0906     2.2027
 Exact  u1    1.1485     0.9764     1.2654     2.0911     2.2027
 Approx u2    0.1370    -0.0250    -0.4107    -0.8577     0.3096
 Exact  u2    0.1366    -0.0266    -0.4100    -0.8567     0.3096

 t =  0.200
 Approx u1    1.0956     1.0529     1.3407     1.8322     2.2035
```

```
Exact  u1    1.0956    1.0515    1.3393    1.8327    2.2050
Approx u2    0.0381    0.1282   -0.7979   -1.1776   -0.4221
Exact  u2    0.0370    0.1247   -0.7961   -1.1784   -0.4221

t =  0.250
Approx u1    0.8119    1.1288    1.5163    1.6076    2.2027
Exact  u1    0.8119    1.1276    1.5142    1.6091    2.2035
Approx u2   -0.4968    0.2123   -1.0259   -1.2149   -1.3938
Exact  u2   -0.4992    0.2078   -1.0257   -1.2183   -1.3938

Number of integration steps in time =     50
Number of function evaluations =   2579
Number of Jacobian evaluations =    20
Number of iterations =    126
```