NAG Library Function Document

nag_pde_parab_1d_coll (d03pdc)

1   Purpose

nag_pde_parab_1d_coll (d03pdc) integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable. The spatial discretization is performed using a Chebyshev C^0 collocation method, and the method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs). The resulting system is solved using a backward differentiation formula method.

2   Specification

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

```
void nag_pde_parab_1d_coll (Integer npde, Integer m, double *ts, double tout,
       void (*pdedef)(Integer npde, double t, const double x[], Integer nptl,
             const double u[], const double ux[], double p[], double q[],
             double r[], Integer *ires, Nag_Comm *comm),
       void (*bndary)(Integer npde, double t, const double u[],
             const double ux[], Integer ibnd, double beta[], double gamma[],
             Integer *ires, Nag_Comm *comm),
       double u[], Integer nbkpts, const double xbkpts[], Integer npoly,
       Integer npts, double x[],
       void (*uinit)(Integer npde, Integer npts, const double x[], double u[],
             Nag_Comm *comm),
       double acc, 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_coll (d03pdc) integrates the system of parabolic equations:

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

where $P_{i,j}$, Q_i and R_i depend on x, t, U, U_x and the vector U is the set of solution values

$$U(x,t) = \left[U_1(x,t), \ldots, U_{\mathbf{npde}}(x,t)\right]^{\mathrm{T}}, \tag{2}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$, Q_i and R_i must not depend on $\dfrac{\partial U}{\partial t}$.

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 $b = x_{\mathbf{nbkpts}}$ are the leftmost and rightmost of a user-defined set of break-points $x_1, x_2, \ldots, x_{\mathbf{nbkpts}}$. The coordinate system in space is defined by the value of m; $m = 0$ for Cartesian coordinates, $m = 1$ for cylindrical polar coordinates and $m = 2$ for spherical polar coordinates.

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

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

The functions R_i, for $i = 1, 2, \ldots, \mathbf{npde}$, which may be thought of as fluxes, are also used in the definition of the boundary conditions for each equation. The boundary conditions must have the form

$$\beta_i(x,t)R_i(x,t,U,U_x) = \gamma_i(x,t,U,U_x), \quad i = 1,2,\ldots,\mathbf{npde}, \tag{3}$$

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

The boundary conditions must be specified in **bndary**. Thus, the problem is subject to the following restrictions:

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

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

(iii) the evaluation of the functions $P_{i,j}$, Q_i and R_i is done at both the break-points and internally selected points for each element in turn, that is $P_{i,j}$, Q_i and R_i are evaluated twice at each break-point. Any discontinuities in these functions **must** therefore be at one or more of the break-points $x_1, x_2, \ldots, x_{\mathbf{nbkpts}}$;

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

(v) if $m > 0$ and $x_1 = 0.0$, which is the left boundary point, then it must be ensured that the PDE solution is bounded at this point. This can be done by either specifying the solution at $x = 0.0$ or by specifying a zero flux there, that is $\beta_i = 1.0$ and $\gamma_i = 0.0$. See also Section 9.

The parabolic equations are approximated by a system of ODEs in time for the values of U_i at the mesh points. This ODE system is obtained by approximating the PDE solution between each pair of break-points by a Chebyshev polynomial of degree **npoly**. The interval between each pair of break-points is treated by nag_pde_parab_1d_coll (d03pdc) as an element, and on this element, a polynomial and its space and time derivatives are made to satisfy the system of PDEs at **npoly** -1 spatial points, which are chosen internally by the code and the break-points. In the case of just one element, the break-points are the boundaries. The user-defined break-points and the internally selected points together define the mesh. The smallest value that **npoly** can take is one, in which case, the solution is approximated by piecewise linear polynomials between consecutive break-points and the method is similar to an ordinary finite element method.

In total there are $(\mathbf{nbkpts} - 1) \times \mathbf{npoly} + 1$ mesh points in the spatial direction, and $\mathbf{npde} \times ((\mathbf{nbkpts} - 1) \times \mathbf{npoly} + 1)$ ODEs in the time direction; one ODE at each break-point for each PDE component and $(\mathbf{npoly} - 1)$ ODEs for each PDE component between each pair of break-points. The system is then integrated forwards in time using a backward differentiation formula 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 and Dew P M (1991) Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs *ACM Trans. Math. Software* **17** 178–206

Zaturska N B, Drazin P G and Banks W H H (1988) On the flow of a viscous fluid driven along a channel by a suction at porous walls *Fluid Dynamics Research* **4**

5   Arguments

1:   **npde** – Integer   *Input*

On entry: the number of PDEs in the system to be solved.

Constraint: $\mathbf{npde} \geq 1$.

2:   **m** – Integer   *Input*

On entry: the coordinate system used:

m = 0
        Indicates Cartesian coordinates.

m $= 1$

Indicates cylindrical polar coordinates.

m $= 2$

Indicates spherical polar coordinates.

Constraint: **m** $= 0$, 1 or 2.

3: **ts** – double * *Input/Output*

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

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

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

4: **tout** – double *Input*

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

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

pdedef must compute the values of the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. The functions may depend on x, t, U and U_x and must be evaluated at a set of points.

The specification of **pdedef** is:

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

1: **npde** – Integer *Input*

On entry: the number of PDEs in the system.

2: **t** – double *Input*

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

3: **x**[**nptl**] – const double *Input*

On entry: contains a set of mesh points at which $P_{i,j}$, Q_i and R_i are to be evaluated. **x**[0] and **x**[**nptl** $- 1$] contain successive user-supplied break-points and the elements of the array will satisfy **x**[0] $<$ **x**[1] $< \cdots <$ **x**[**nptl** $- 1$].

4: **nptl** – Integer *Input*

On entry: the number of points at which evaluations are required (the value of **npoly** $+ 1$).

5: **u**[**npde** $\times$ **nptl**] – const double *Input*

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

6: **ux**[**npde** $\times$ **nptl**] – const double *Input*

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

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

On exit: **p**[**npde** $\times$ **npde** $\times (k-1) +$ **npde** $\times (j-1) + (i-1)$] must be set to the value of $P_{i,j}(x, t, U, U_x)$ where $x = $ **x**[$k-1$], for $i = 1, 2, \ldots,$ **npde**, $j = 1, 2, \ldots,$ **npde** and $k = 1, 2, \ldots,$ **nptl**.

8:    **q**[**npde** $\times$ **nptl**] – double   *Output*

On exit: **q**[**npde** $\times (j-1) + i - 1$] must be set to the value of $Q_i(x, t, U, U_x)$ where $x = $ **x**[$j-1$], for $i = 1, 2, \ldots,$ **npde** and $j = 1, 2, \ldots,$ **nptl**.

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

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

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

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, you may set **ires** to force the integration 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_coll (d03pdc) returns to the calling function with the error indicator set to **fail**.**code** = NE_FAILED_DERIV.

11:   **comm** – Nag_Comm *

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_coll (d03pdc) you may allocate memory and initialize these pointers with various quantities for use by **pdedef** when called from nag_pde_parab_1d_coll (d03pdc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

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

bndary must compute the functions β_i and γ_i which define the boundary conditions as in equation (3).

The specification of **bndary** is:

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

1:    **npde** – Integer   *Input*

On entry: the number of PDEs in the system.

2:    **t** – double   *Input*

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

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

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

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

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

5:     **ibnd** – Integer   *Input*

On entry: specifies which boundary conditions are to be evaluated.

ibnd $= 0$
       bndary must set up the coefficients of the left-hand boundary, $x = a$.

ibnd $\neq 0$
       bndary must set up the coefficients of the right-hand boundary, $x = b$.

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

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

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

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

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

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, you may set **ires** to force the integration 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_coll (d03pdc) returns to the calling function with the error indicator set to **fail.code** = NE_FAILED_DERIV.

9:     **comm** – Nag_Comm *

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_coll (d03pdc) you may allocate memory and initialize these pointers with various quantities for use by **bndary** when called from nag_pde_parab_1d_coll (d03pdc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

7:     **u[npde × npts]** – double   *Input/Output*

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

On exit: $\mathbf{u}[\mathbf{npde} \times (j-1) + i - 1]$ will contain the computed solution at $t = \mathbf{ts}$.

8:    **nbkpts** – Integer   *Input*

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

Constraint: $\mathbf{nbkpts} \geq 2$.

9:    **xbkpts**[**nbkpts**] – const double   *Input*

On entry: the values of the break-points in the space direction. **xbkpts**[0] must specify the left-hand boundary, a, and **xbkpts**[**nbkpts** $- 1$] must specify the right-hand boundary, b.

Constraint: $\mathbf{xbkpts}[0] < \mathbf{xbkpts}[1] < \cdots < \mathbf{xbkpts}[\mathbf{nbkpts} - 1]$.

10:   **npoly** – Integer   *Input*

On entry: the degree of the Chebyshev polynomial to be used in approximating the PDE solution between each pair of break-points.

Constraint: $1 \leq \mathbf{npoly} \leq 49$.

11:   **npts** – Integer   *Input*

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

Constraint: $\mathbf{npts} = (\mathbf{nbkpts} - 1) \times \mathbf{npoly} + 1$.

12:   **x**[**npts**] – double   *Output*

On exit: the mesh points chosen by nag_pde_parab_1d_coll (d03pdc) in the spatial direction. The values of **x** will satisfy $\mathbf{x}[0] < \mathbf{x}[1] < \cdots < \mathbf{x}[\mathbf{npts} - 1]$.

13:   **uinit** – function, supplied by the user   *External Function*

uinit must compute the initial values of the PDE components $U_i(x_j, t_0)$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{npts}$.

The specification of **uinit** is:

```
void uinit (Integer npde, Integer npts, const double x[], double u[],
     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:    **x**[**npts**] – const double   *Input*

On entry: $\mathbf{x}[j - 1]$, contains the values of the jth mesh point, for $j = 1, 2, \ldots, \mathbf{npts}$.

4:    **u**[**npde** × **npts**] – double   *Output*

On exit: $\mathbf{u}[\mathbf{npde} \times (j-1) + i - 1]$ must be set to the initial value $U_i(x_j, t_0)$, for $i = 1, 2, \ldots, \mathbf{npde}$ and $j = 1, 2, \ldots, \mathbf{npts}$.

5:    **comm** – Nag_Comm *

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

> **user** – double *
> **iuser** – Integer *
> **p** – Pointer
>
>> The type Pointer will be void *. Before calling nag_pde_parab_1d_coll (d03pdc) you may allocate memory and initialize these pointers with various quantities for use by **uinit** when called from nag_pde_parab_1d_coll (d03pdc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

14: **acc** – double *Input*

On entry: a positive quantity for controlling the local error estimate in the time integration. If $E(i, j)$ is the estimated error for U_i at the jth mesh point, the error test is:

$$|E(i, j)| = \mathbf{acc} \times (1.0 + |\mathbf{u}[\mathbf{npde} \times (j - 1) + i - 1]|).$$

Constraint: $\mathbf{acc} > 0.0$.

15: **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.

16: **lrsave** – Integer *Input*

On entry: the dimension of the array **rsave**.

Constraint: $\mathbf{lrsave} \geq 11 \times \mathbf{npde} \times \mathbf{npts} + 50 + nwkres + lenode$.

17: **isave**[**lisave**] – Integer *Communication Array*

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

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

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

isave[1]
> Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves computing the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

isave[2]
> Contains the number of Jacobian evaluations performed by the time integrator.

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

isave[4]
> Contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the LU decomposition of the Jacobian matrix.

18: **lisave** – Integer *Input*

On entry: the dimension of the array **isave**.

Constraint: $\mathbf{lisave} \geq \mathbf{npde} \times \mathbf{npts} + 24$.

19:   **itask** – Integer   *Input*

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

itask = 1
        Normal computation of output values **u** at $t = $ **tout**.

itask = 2
        One step and return.

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

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

20:   **itrace** – Integer   *Input*

On entry: the level of trace information required from nag_pde_parab_1d_coll (d03pdc) and the underlying ODE solver. **itrace** may take the value -1, 0, 1, 2 or 3.

itrace = -1
        No output is generated.

itrace = 0
        Only warning messages from the PDE solver are printed.

itrace > 0
        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.

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

The advisory messages are given in greater detail as **itrace** increases.

21:   **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.

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

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

ind = 0
        Starts or restarts the integration in time.

ind = 1
        Continues the integration after an earlier exit from the function. In this case, only the arguments **tout** and **fail** should be reset between calls to nag_pde_parab_1d_coll (d03pdc).

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

On exit: **ind** = 1.

23:   **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

24:   **saved** – Nag_D03_Save *   *Communication Structure*

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

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

    The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6   Error Indicators and Warnings

NE_ACC_IN_DOUBT

    Integration completed, but a small change in **acc** is unlikely to result in a changed solution. **acc** $= \langle value \rangle$.

NE_ALLOC_FAIL

    Dynamic memory allocation failed.
    See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

    On entry, argument $\langle value \rangle$ 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

    acc was too small to start integration: **acc** $= \langle value \rangle$.

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** $= \langle value \rangle$.

    Underlying ODE solver cannot make further progress from the point **ts** with the supplied value of **acc**. **ts** $= \langle value \rangle$, **acc** $= \langle value \rangle$.

NE_INCOMPAT_PARAM

    On entry, **m** $= \langle value \rangle$ and **xbkpts**[0] $= \langle value \rangle$.
    Constraint: **m** ≤ 0 or **xbkpts**[0] ≥ 0.0

NE_INT

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

    On entry, **ind** $= \langle value \rangle$.
    Constraint: **ind** $= 0$ or 1.

    On entry, **itask** $= \langle value \rangle$.
    Constraint: **itask** $= 1, 2$ or 3.

    On entry, **m** $= \langle value \rangle$.
    Constraint: **m** $= 0, 1$ or 2.

    On entry, **nbkpts** $= \langle value \rangle$.
    Constraint: **nbkpts** ≥ 2.

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

On entry, **npoly** = ⟨*value*⟩.
Constraint: $1 \le$ **npoly** ≤ 49.

On entry, **npoly** = ⟨*value*⟩.
Constraint: **npoly** ≤ 49.

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

NE_INT_2

On entry, **lisave** is too small: **lisave** = ⟨*value*⟩. Minimum possible dimension: ⟨*value*⟩.

On entry, **lrsave** is too small: **lrsave** = ⟨*value*⟩. Minimum possible dimension: ⟨*value*⟩.

NE_INT_3

On entry, **npts** = ⟨*value*⟩, **nbkpts** = ⟨*value*⟩ and **npoly** = ⟨*value*⟩.
Constraint: **npts** = (**nbkpts** − 1) × **npoly** + 1.

NE_INTERNAL_ERROR

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

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

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

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*value*⟩.

NE_NOT_STRICTLY_INCREASING

On entry, break-points **xbkpts** are badly ordered: $I = $⟨*value*⟩, **xbkpts**[$I − 1$] = ⟨*value*⟩, $J = $⟨*value*⟩ and **xbkpts**[$J − 1$] = ⟨*value*⟩.

NE_NOT_WRITE_FILE

Cannot open file ⟨*value*⟩ for writing.

NE_REAL

On entry, **acc** = ⟨*value*⟩.
Constraint: **acc** > 0.0.

NE_REAL_2

On entry, **tout** = ⟨*value*⟩ and **ts** = ⟨*value*⟩.
Constraint: **tout** $>$ **ts**.

On entry, **tout** − **ts** is too small: **tout** = ⟨*value*⟩ and **ts** = ⟨*value*⟩.

NE_SING_JAC

Singular Jacobian of ODE system. Check problem formulation.

NE_TIME_DERIV_DEP

Flux function appears to depend on time derivatives.

NE_USER_STOP

In evaluating residual of ODE system, **ires** = 2 has been set in **pdedef** or **bndary**. Integration is successful as far as **ts**: **ts** = ⟨*value*⟩.

7   Accuracy

nag_pde_parab_1d_coll (d03pdc) controls the accuracy of the integration in the time direction but not the accuracy of the approximation in space. The spatial accuracy depends on the degree of the polynomial approximation **npoly**, and on both the number of break-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 argument, **acc**.

8   Parallelism and Performance

nag_pde_parab_1d_coll (d03pdc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_pde_parab_1d_coll (d03pdc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Notefor your implementation for any additional implementation-specific information.

9   Further Comments

nag_pde_parab_1d_coll (d03pdc) is designed to solve parabolic systems (possibly including elliptic equations) with second-order derivatives in space. The argument specification allows you to include equations with only first-order derivatives in the space direction but there is no guarantee that the method of integration will be satisfactory for such systems. The position and nature of the boundary conditions in particular are critical in defining a stable problem.

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

10   Example

The problem consists of a fourth-order PDE which can be written as a pair of second-order elliptic-parabolic PDEs for $U_1(x,t)$ and $U_2(x,t)$,

$$0 = \frac{\partial^2 U_1}{\partial x^2} - U_2 \tag{4}$$

$$\frac{\partial U_2}{\partial t} = \frac{\partial^2 U_2}{\partial x^2} + U_2 \frac{\partial U_1}{\partial x} - U_1 \frac{\partial U_2}{\partial x} \tag{5}$$

where $-1 \leq x \leq 1$ and $t \geq 0$. The boundary conditions are given by

$$\frac{\partial U_1}{\partial x} = 0 \quad \text{and} \quad U_1 = 1 \quad \text{at } x = -1, \quad \text{and}$$

$$\frac{\partial U_1}{\partial x} = 0 \quad \text{and} \quad U_1 = -1 \quad \text{at } x = 1.$$

The initial conditions at $t = 0$ are given by

$$U_1 = -\sin \frac{\pi x}{2} \quad \text{and} \quad U_2 = \frac{\pi^2}{4} \sin \frac{\pi x}{2}.$$

The absence of boundary conditions for $U_2(x,t)$ does not pose any difficulties provided that the

derivative flux boundary conditions are assigned to the first PDE (4) which has the correct flux, $\frac{\partial U_1}{\partial x}$. The conditions on $U_1(x,t)$ at the boundaries are assigned to the second PDE by setting $\beta_2 = 0.0$ in equation (3) and placing the Dirichlet boundary conditions on $U_1(x,t)$ in the function γ_2.

10.1 Program Text

```c
/* nag_pde_parab_1d_coll (d03pdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd03.h>
#include <nagx01.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL uinit(Integer, Integer, const double[], double[],
                             Nag_Comm *);
  static void NAG_CALL pdedef(Integer, double, const double[], Integer,
                              const double[], const double[], double[],
                              double[], double[], Integer *, Nag_Comm *);
  static void NAG_CALL bndary(Integer, double, const double[], const double[],
                              Integer, double[], double[], Integer *,
                              Nag_Comm *);
#ifdef __cplusplus
}
#endif

#define U(I, J)        u[npde*((J) -1)+(I) -1]
#define UOUT(I, J, K)  uout[npde*(intpts*((K) -1)+(J) -1)+(I) -1]
#define P(I, J, K)     p[npde*(npde*((K) -1)+(J) -1)+(I) -1]
#define Q(I, J)        q[npde*((J) -1)+(I) -1]
#define R(I, J)        r[npde*((J) -1)+(I) -1]
#define UX(I, J)       ux[npde*((J) -1)+(I) -1]

int main(void)
{
  const Integer nbkpts = 10, nelts = nbkpts - 1, npde = 2, npoly = 3,
        m = 0, itype = 1, npts = nelts * npoly + 1, neqn = npde * npts,
        intpts = 6, npl1 = npoly + 1, lisave = neqn + 24,
        mu = npde * (npoly + 1) - 1, lenode = (3 * mu + 1) * neqn,
        nwkres =
        3 * npl1 * npl1 + npl1 * (npde * npde + 6 * npde + nbkpts + 1)
        + 13 * npde + 5, lrsave = 11 * neqn + 50 + nwkres + lenode;

  static double ruser[3] = { -1.0, -1.0, -1.0 };
  static double xout[6] = { -1., -.6, -.2, .2, .6, 1. };
  double acc, tout, ts;
  Integer exit_status = 0, i, ind, it, itask, itrace;

  double *rsave = 0, *u = 0, *uout = 0, *x = 0, *xbkpts = 0;
  Integer *isave = 0;
  NagError fail;
  Nag_Comm comm;
  Nag_D03_Save saved;

  INIT_FAIL(fail);
```

```
  printf("nag_pde_parab_1d_coll (d03pdc) Example Program Results\n\n");

/* For communication with user-supplied functions: */
comm.user = ruser;

/* Allocate memory */

if (!(rsave = NAG_ALLOC(lrsave, double)) ||
    !(u = NAG_ALLOC(npde * npts, double)) ||
    !(uout = NAG_ALLOC(npde * intpts * itype, double)) ||
    !(x = NAG_ALLOC(npts, double)) ||
    !(xbkpts = NAG_ALLOC(nbkpts, double)) ||
    !(isave = NAG_ALLOC(lisave, Integer)))
{
  printf("Allocation failure\n");
  exit_status = 1;
  goto END;
}

acc = 1e-4;
itrace = 0;

/* Set the break-points */

for (i = 0; i < 10; ++i) {
  xbkpts[i] = i * 2.0 / 9.0 - 1.0;
}

ind = 0;
itask = 1;
ts = 0.0;
tout = 1e-5;
printf(" Polynomial degree =%4" NAG_IFMT "", npoly);
printf("   No. of elements = %4" NAG_IFMT "\n\n", nelts);
printf(" Accuracy requirement = %12.3e", acc);
printf("   Number of points = %5" NAG_IFMT "\n\n", npts);
printf("   t /   x    ");

for (i = 0; i < 6; ++i) {
  printf("%8.4f", xout[i]);
  printf((i + 1) % 6 == 0 || i == 5 ? "\n" : "");
}
printf("\n");

/* Loop over output values of t */

for (it = 0; it < 5; ++it) {
  tout *= 10.0;

  /* nag_pde_parab_1d_coll (d03pdc).
   * General system of parabolic PDEs, method of lines,
   * Chebyshev C^0 collocation, one space variable
   */
  nag_pde_parab_1d_coll(npde, m, &ts, tout, pdedef, bndary, u, nbkpts,
                        xbkpts, npoly, npts, x, uinit, acc, rsave, lrsave,
                        isave, lisave, itask, itrace, 0, &ind, &comm,
                        &saved, &fail);

  if (fail.code != NE_NOERROR) {
    printf("Error from nag_pde_parab_1d_coll (d03pdc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Interpolate at required spatial points */

  /* nag_pde_interp_1d_coll (d03pyc).
   * PDEs, spatial interpolation with nag_pde_parab_1d_coll
   * (d03pdc) or nag_pde_parab_1d_coll_ode (d03pjc)
   */
```

```
    nag_pde_interp_1d_coll(npde, u, nbkpts, xbkpts, npoly, npts, xout,
                           intpts, itype, uout, rsave, lrsave, &fail);

    if (fail.code != NE_NOERROR) {
      printf("Error from nag_pde_interp_1d_coll (d03pyc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

    printf("\n %6.4f u(1)", tout);

    for (i = 1; i <= 6; ++i) {
      printf("%8.4f", UOUT(1, i, 1));
      printf(i % 6 == 0 || i == 6 ? "\n" : "");
    }

    printf("        u(2)");

    for (i = 1; i <= 6; ++i) {
      printf("%8.4f", UOUT(2, i, 1));
      printf(i % 6 == 0 || i == 6 ? "\n" : "");
    }
  }

  /* Print integration statistics */

  printf("\n");
  printf(" Number of integration steps in time                ");
  printf("%4" NAG_IFMT "\n", isave[0]);
  printf(" Number of residual evaluations of resulting ODE system ");
  printf("%4" NAG_IFMT "\n", isave[1]);
  printf(" Number of Jacobian evaluations                     ");
  printf("%4" NAG_IFMT "\n", isave[2]);
  printf(" Number of iterations of nonlinear solver           ");
  printf("%4" NAG_IFMT "\n", isave[4]);

END:
  NAG_FREE(rsave);
  NAG_FREE(u);
  NAG_FREE(uout);
  NAG_FREE(x);
  NAG_FREE(xbkpts);
  NAG_FREE(isave);

  return exit_status;
}

static void NAG_CALL uinit(Integer npde, Integer npts, const double x[],
                           double u[], Nag_Comm *comm)
{
  Integer i;
  double piby2;

  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback uinit, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  piby2 = 0.5 * nag_pi;
  for (i = 1; i <= npts; ++i) {
    U(1, i) = -sin(piby2 * x[i - 1]);
    U(2, i) = -piby2 * piby2 * U(1, i);
  }
  return;
}

static void NAG_CALL pdedef(Integer npde, double t, const double x[],
                            Integer nptl, const double u[], const double ux[],
                            double p[], double q[], double r[], Integer *ires,
                            Nag_Comm *comm)
{
```

```
    Integer i;

    if (comm->user[1] == -1.0) {
      printf("(User-supplied callback pdedef, first invocation.)\n");
      comm->user[1] = 0.0;
    }
    for (i = 1; i <= nptl; ++i) {
      Q(1, i) = U(2, i);
      Q(2, i) = U(1, i) * UX(2, i) - UX(1, i) * U(2, i);
      R(1, i) = UX(1, i);
      R(2, i) = UX(2, i);
      P(1, 1, i) = 0.0;
      P(1, 2, i) = 0.0;
      P(2, 1, i) = 0.0;
      P(2, 2, i) = 1.0;
    }
    return;
}

static void NAG_CALL bndary(Integer npde, double t, const double u[],
                            const double ux[], Integer ibnd, double beta[],
                            double gamma[], Integer *ires, Nag_Comm *comm)
{
    if (comm->user[2] == -1.0) {
      printf("(User-supplied callback bndary, first invocation.)\n");
      comm->user[2] = 0.0;
    }
    if (ibnd == 0) {
      beta[0] = 1.0;
      gamma[0] = 0.0;
      beta[1] = 0.0;
      gamma[1] = u[0] - 1.0;
    }
    else {
      beta[0] = 1.0;
      gamma[0] = 0.0;
      beta[1] = 0.0;
      gamma[1] = u[0] + 1.0;
    }
    return;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_pde_parab_1d_coll (d03pdc) Example Program Results

 Polynomial degree =   3   No. of elements =     9

 Accuracy requirement =    1.000e-04  Number of points =    28

  t /   x    -1.0000 -0.6000 -0.2000  0.2000  0.6000  1.0000

(User-supplied callback uinit, first invocation.)
(User-supplied callback pdedef, first invocation.)
(User-supplied callback bndary, first invocation.)

 0.0001 u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
        u(2) -2.4850 -1.9957 -0.7623  0.7623  1.9957  2.4850

 0.0010 u(1)  1.0000  0.8085  0.3088 -0.3088 -0.8085 -1.0000
        u(2) -2.5583 -1.9913 -0.7606  0.7606  1.9913  2.5583

 0.0100 u(1)  1.0000  0.8051  0.3068 -0.3068 -0.8051 -1.0000
        u(2) -2.6962 -1.9481 -0.7439  0.7439  1.9481  2.6962

 0.1000 u(1)  1.0000  0.7951  0.2985 -0.2985 -0.7951 -1.0000
```

```
      u(2) -2.9022 -1.8339 -0.6338  0.6338  1.8339  2.9022

 1.0000 u(1)  1.0000  0.7939  0.2972 -0.2972 -0.7939 -1.0000
       u(2) -2.9233 -1.8247 -0.6120  0.6120  1.8247  2.9233

Number of integration steps in time                      50
Number of residual evaluations of resulting ODE system  407
Number of Jacobian evaluations                           18
Number of iterations of nonlinear solver                122
```
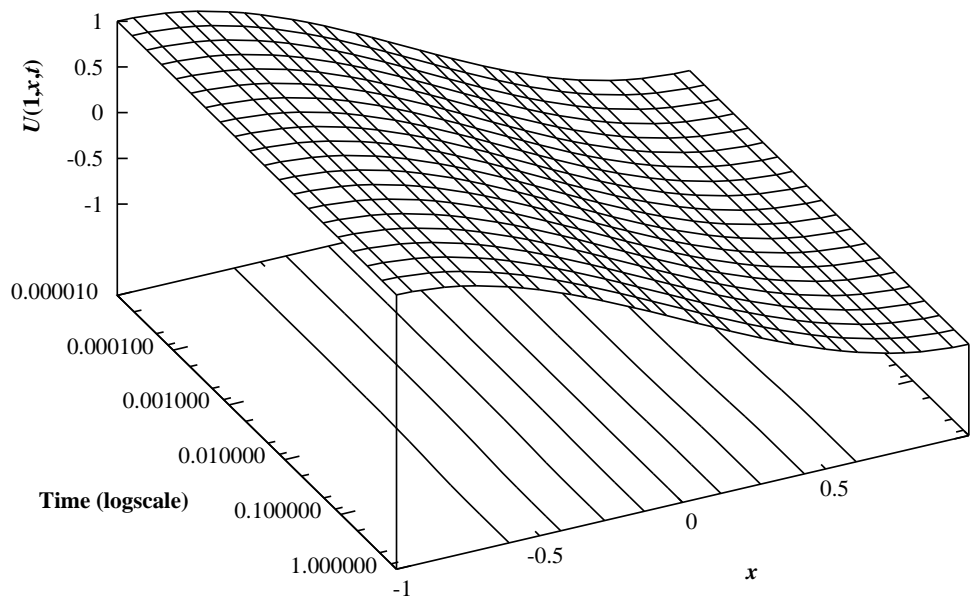
Example Program
Solution, $U(1,x,t)$, of Elliptic-parabolic Pair using Chebyshev Collocation and BDF

Solution, $U(2,x,t)$, of Elliptic-parabolic Pair using Chebyshev Collocation and BDF