# NAG Toolbox

# nag_pde_1d_parab_coll (d03pd)

## 1 Purpose

nag_pde_1d_parab_coll (d03pd) 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 Syntax

```
[ts, u, x, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] =
nag_pde_1d_parab_coll(m, ts, tout, pdedef, bndary, u, xbkpts, npoly, uinit, acc,
rsave, isave, itask, itrace, ind, cwsav, lwsav, iwsav, rwsav, 'npde', npde,
'nbkpts', nbkpts, 'npts', npts, 'user', user)
```

```
[ts, u, x, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] = d03pd
(m, ts, tout, pdedef, bndary, u, xbkpts, npoly, uinit, acc, rsave, isave,
itask, itrace, ind, cwsav, lwsav, iwsav, rwsav, 'npde', npde, 'nbkpts', nbkpts,
'npts', npts, 'user', user)
```

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

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

## 3 Description

nag_pde_1d_parab_coll (d03pd) integrates the system of parabolic equations:

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

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 $t_{\mathrm{out}}$, over the space interval $a \le x \le b$, 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_1d_parab_coll (d03pd) 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 Parameters

## 5.1 Compulsory Input Parameters

1: **m** – INTEGER

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.

2: **ts** – REAL (KIND=nag_wp)

The initial value of the independent variable $t$.

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

3: **tout** – REAL (KIND=nag_wp)

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

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

**pdedef** must 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.

```
[p, q, r, ires, user] = pdedef(npde, t, x, nptl, u, ux, ires, user)
```

**Input Parameters**

1: **npde** – INTEGER

The number of PDEs in the system.

2: **t** – REAL (KIND=nag_wp)

The current value of the independent variable $t$.

3: **x**(**nptl**) – REAL (KIND=nag_wp) array

Contains a set of mesh points at which $P_{i,j}$, $Q_i$ and $R_i$ are to be evaluated. $\mathbf{x}(1)$ and $\mathbf{x}(\mathbf{nptl})$ contain successive user-supplied break-points and the elements of the array will satisfy $\mathbf{x}(1) < \mathbf{x}(2) < \cdots < \mathbf{x}(\mathbf{nptl})$.

4: **nptl** – INTEGER

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

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

$\mathbf{u}(i,j)$ contains the value of the component $U_i(x,t)$ where $x = \mathbf{x}(j)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,\ldots,$ **nptl**.

6: **ux**(**npde**, **nptl**) – REAL (KIND=nag_wp) array

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

7: **ires** – INTEGER

Set to $-1$ or 1.

8: **user** – INTEGER array

**pdedef** is called from nag_pde_1d_parab_coll (d03pd) with the object supplied to nag_pde_1d_parab_coll (d03pd).

**Output Parameters**

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

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

2: **q**(**npde**, **nptl**) – REAL (KIND=nag_wp) array

**q**$(i,j)$ must be set to the value of $Q_i(x,t,U,U_x)$ where $x = \mathbf{x}(j)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,\ldots,$ **nptl**.

3: **r**(**npde**, **nptl**) – REAL (KIND=nag_wp) array

**r**$(i,j)$ must be set to the value of $R_i(x,t,U,U_x)$ where $x = \mathbf{x}(j)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,\ldots,$ **nptl**.

4: **ires** – INTEGER

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

**ires** $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)routine with the error indicator set to **ifail** $= 6$.

**ires** $= 3$
Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set **ires** $= 3$, then nag_pde_1d_parab_coll (d03pd) returns to the calling function with the error indicator set to **ifail** $= 4$.

5: **user** – INTEGER array

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

**bndary** must compute the functions $\beta_i$ and $\gamma_i$ which define the boundary conditions as in equation (3).

```
[beta, gamma, ires, user] = bndary(npde, t, u, ux, ibnd, ires, user)
```

**Input Parameters**

1: **npde** – INTEGER

The number of PDEs in the system.

2: **t** – REAL (KIND=nag_wp)

The current value of the independent variable $t$.

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

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

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

**ux**$(i)$ 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

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: **ires** – INTEGER

Set to $-1$ or $1$.

7: **user** – INTEGER array

**bndary** is called from nag_pde_1d_parab_coll (d03pd) with the object supplied to nag_pde_1d_parab_coll (d03pd).

**Output Parameters**

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

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

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

**gamma**$(i)$ 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**.

3: **ires** – INTEGER

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

**ires** $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)routine with the error indicator set to **ifail** $= 6$.

**ires** $= 3$
Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set **ires** $= 3$ when a physically meaningless input or output value has been generated. If you consecutively set **ires** $= 3$, then nag_pde_1d_parab_coll (d03pd) returns to the calling function with the error indicator set to **ifail** $= 4$.

4: **user** – INTEGER array

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

If **ind** $= 1$ the value of **u** must be unchanged from the previous call.

7: **xbkpts**(**nbkpts**) – REAL (KIND=nag_wp) array

The values of the break-points in the space direction. **xbkpts**$(1)$ must specify the left-hand boundary, $a$, and **xbkpts**(**nbkpts**) must specify the right-hand boundary, $b$.

*Constraint*: **xbkpts**$(1) <$ **xbkpts**$(2) < \cdots <$ **xbkpts**(**nbkpts**).

8: **npoly** – INTEGER

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

*Constraint*: $1 \leq$ **npoly** $\leq 49$.

9: **uinit** – SUBROUTINE, supplied by the user.

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

```
[u, user] = uinit(npde, npts, x, user)
```

**Input Parameters**

1: **npde** – INTEGER

The number of PDEs in the system.

2: **npts** – INTEGER

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

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

$\mathbf{x}(j)$, contains the values of the $j$th mesh point, for $j = 1,2,\ldots,$ **npts**.

4: **user** – INTEGER array

**uinit** is called from nag_pde_1d_parab_coll (d03pd) with the object supplied to nag_pde_1d_parab_coll (d03pd).

**Output Parameters**

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

$\mathbf{u}(i,j)$ must be set to the initial value $U_i(x_j,t_0)$, for $i = 1,2,\ldots,$ **npde** and $j = 1,2,\ldots,$ **npts**.

2: **user** – INTEGER array

10: **acc** – REAL (KIND=nag_wp)

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 $j$th mesh point, the error test is:

$$|E(i,j)| = \mathbf{acc} \times (1.0 + |\mathbf{u}(i,j)|).$$

*Constraint*: **acc** $> 0.0$.

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

$lrsave$, the dimension of the array, must satisfy the constraint $lrsave \geq 11 \times$ **npde** $\times$ **npts** $+ 50 + nwkres + lenode$.

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.

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

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

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

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

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

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

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

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

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

13: **itask** – INTEGER

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.

14: **itrace** – INTEGER

The level of trace information required from nag_pde_1d_parab_coll (d03pd) and the underlying ODE solver. **itrace** may take the value −1, 0, 1, 2 or 3.

**itrace** = −1
No output is generated.

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

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

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

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

15: **ind** – INTEGER

Indicates whether this is a continuation call or a new integration.

**ind** = 0
Starts or restarts the integration in time.

**ind** = 1
Continues the integration after an earlier exit from the function. In this case, only the arguments **tout** and **ifail** should be reset between calls to nag_pde_1d_parab_coll (d03pd).

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

16: **cwsav**(**10**) – CHARACTER(80) array

17: **lwsav**(**100**) – LOGICAL array

18: **iwsav**(**505**) – INTEGER array

19: **rwsav**(**1100**) – REAL (KIND=nag_wp) array

## 5.2 Optional Input Parameters

1: **npde** – INTEGER

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

The number of PDEs in the system to be solved.

*Constraint*: **npde** $\geq 1$.

2: **nbkpts** – INTEGER

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

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

*Constraint*: **nbkpts** $\geq 2$.

3: **npts** – INTEGER

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

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

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

4: **user** – INTEGER array

**user** is not used by nag_pde_1d_parab_coll (d03pd), but is passed to **pdedef**, **bndary** and **uinit**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

## 5.3 Output Parameters

1: **ts** – REAL (KIND=nag_wp)

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

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

**u**$(i, j)$ will contain the computed solution at $t =$ **ts**.

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

The mesh points chosen by nag_pde_1d_parab_coll (d03pd) in the spatial direction. The values of **x** will satisfy **x**$(1) <$ **x**$(2) < \cdots <$ **x**(**npts**).

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

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

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

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

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

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

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

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

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

6: **ind** – INTEGER

**ind** $= 1$.

7: **user** – INTEGER array

8: **cwsav**(**10**) – CHARACTER(80) array

9: **lwsav**(**100**) – LOGICAL array

10: **iwsav**(**505**) – INTEGER array

11: **rwsav**(**1100**) – REAL (KIND=nag_wp) array

12: **ifail** – INTEGER

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

## 6 Error Indicators and Warnings

Errors or warnings detected by the function:

**ifail** $= 1$

On entry, **tout** $\leq$ **ts**,
or **tout** $-$ **ts** is too small,
or **itask** $\neq 1$, 2 or 3,
or **m** $\neq 0$, 1 or 2,
or **m** $> 0$ and **xbkpts**$(1) < 0.0$,
or **npde** $< 1$,
or **nbkpts** $< 2$,
or **npoly** $< 1$ or **npoly** $> 49$,
or **npts** $\neq$ (**nbkpts** $- 1$) $\times$ **npoly** $+ 1$,
or **acc** $\leq 0.0$,
or **ind** $\neq 0$ or 1,
or break-points **xbkpts**$(i)$ are not ordered,
or $lrsave$ is too small,
or $lisave$ is too small.

**ifail** $= 2$ (*warning*)

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

**ifail** $= 3$ (*warning*)

In the underlying ODE solver, there were repeated errors or corrector convergence test failures on an attempted step, before completing the requested task. The problem may have a singularity or **acc** is too small for the integration to continue. Integration was successful as far as $t =$ **ts**.

**ifail** $= 4$

In setting up the ODE system, the internal initialization function was unable to initialize the derivative of the ODE system. This could be due to the fact that **ires** was repeatedly set to 3 in at least **pdedef** or **bndary**, when the residual in the underlying ODE solver was being evaluated.

**ifail** $= 5$

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

**ifail** $= 6$ (*warning*)

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

**ifail** $= 7$

The value of **acc** is so small that the function is unable to start the integration in time.

**ifail** $= 8$

In one of **pdedef** or **bndary**, **ires** was set to an invalid value.

**ifail** $= 9$ (nag_ode_ivp_stiff_imp_revcom (d02nn))

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

**ifail** $= 10$ (*warning*)

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

**ifail** $= 11$

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

**ifail** $= 12$

Not applicable.

**ifail** $= 13$

Not applicable.

**ifail** $= 14$

The flux function $R_i$ was detected as depending on time derivatives, which is not permissible.

**ifail** $= -99$

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

**ifail** $= -399$

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

**ifail** $= -999$

Dynamic memory allocation failed.

## 7 Accuracy

nag_pde_1d_parab_coll (d03pd) 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 Further Comments

nag_pde_1d_parab_coll (d03pd) 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.

## 9 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 $\gamma_2$.

### 9.1 Program Text

```
function d03pd_example

fprintf('d03pd example results\n\n');

% Solution of an elliptic-parabolic pair of PDEs
% (derived from a fourth-order PDE).

% Set values for problem parameters.
npde = 2;

% Number of points on interpolated mesh, number of break points.
intpts = 6;
nbkpts = 10;
```

```
% Order of Chebyshev polynomial.
npoly = nag_int(3);

npts   = (nbkpts-1)*npoly + 1;
lisave = npde*npts + 24;

% Define some arrays.
rsave   = zeros(4000, 1);
u       = zeros(npde, npts);
isave   = zeros(lisave, 1, nag_int_name);
lwsav   = false(100, 1);
iwsav   = zeros(505, 1, nag_int_name);
rwsav   = zeros(1100, 1);
cwsav   = {'', '', '', '', '', '', '', '', '', ''};
% Set up the points on the interpolation grid.
xinterp = [-1:0.4:1];

% Number of output points in time.
niter  = 20;

% Prepare to store plotting results.
tsav   = zeros(niter, 1);
usav   = zeros(2, niter, npts);
isav   = 0;

% Set the break points.
hx     = 2/(nbkpts-1);
xbkpts = [-1:hx:1];

% Set initial conditions.
m      = nag_int(0);
ts     = 0.0;
tout   = 0.1e-4;
acc    = 1.0e-4;
itask  = nag_int(1);
itrace = nag_int(0);
ind0   = nag_int(0);

alpha  = -log(tout)/(niter-1);

% Output algorithmic details and interpolation points.
fprintf('polynomial degree = %4d    no. of elements = %4d\n', npoly, nbkpts-1);
fprintf('accuracy requirement = %10.3e    number of points = %5d', acc, npts);
fprintf('\n\n t / x       ');
fprintf('%8.4f', xinterp);
fprintf('\n\n');

% Loop over exponentially increasing endpoints of integration 0.0001 --> 1.
% Set itask = 1: normal computation of output values at t = tout.
for iter = 1:niter

  tout = exp(alpha*(iter - niter));

  % (re)start integration in time: ind0=0.
  [ts, u, x, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] = ...
  d03pd( ...
         m, ts, tout, @pdedef, ...
         @bndary, u, xbkpts, npoly, @uinit, ...
         acc, rsave, isave, itask, itrace, ind0, ...
         cwsav, lwsav, iwsav, rwsav);

  % Interpolation onto coarser grid for display
  itype  = nag_int(1);
  [uinterp, rsave, ifail] = d03py( ...
                                  u, xbkpts, npoly, xinterp, itype, rsave);

  % Output interpolated results for this time step.
  fprintf('%7.4f  u(1)', ts);
  fprintf('%8.4f', uinterp(1,:,1));
  fprintf('\n         u(2)');
```

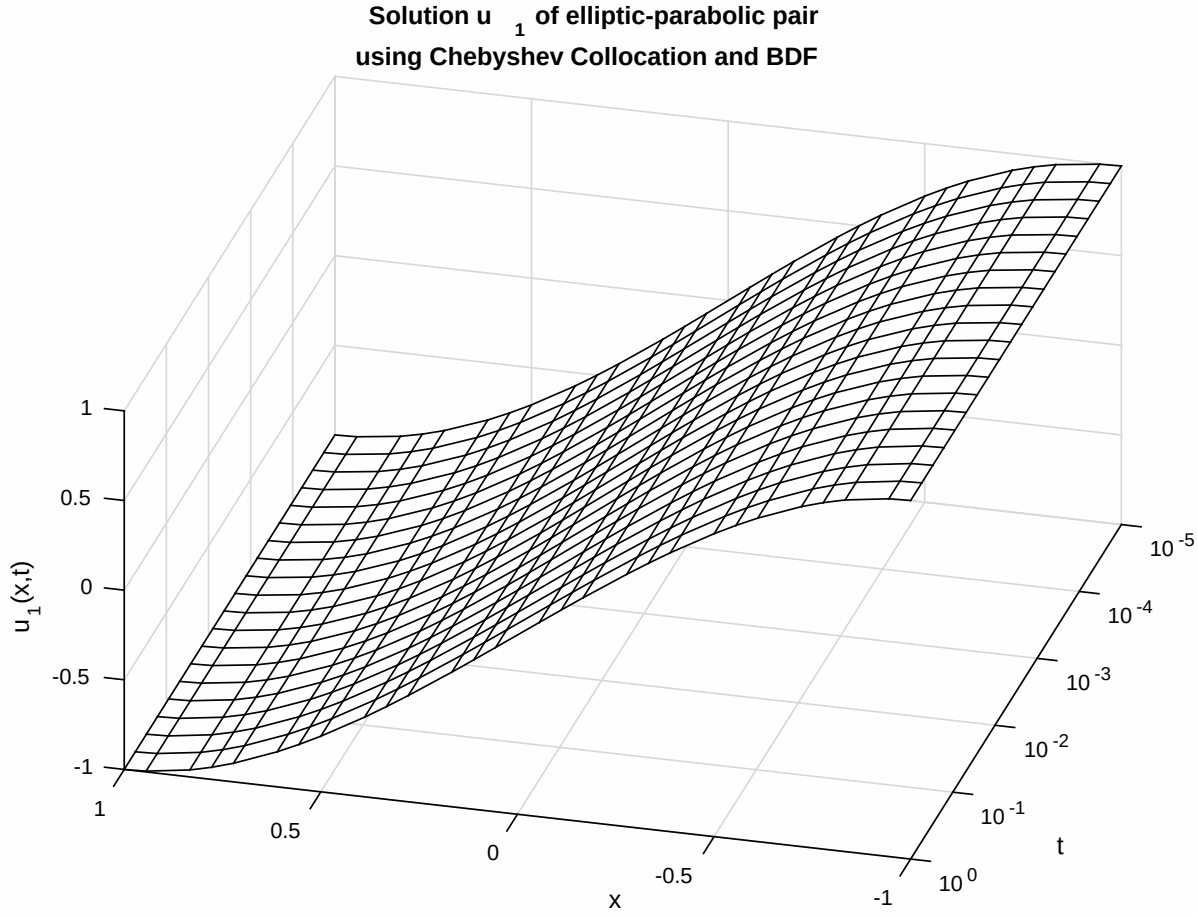
```
  fprintf('%8.4f', uinterp(2,:,1));
  fprintf('\n\n');

  % Save this timestep for plotting.
  isav = isav+1;
  tsav(isav) = ts;
  usav(1:2,isav,1:npts) = u(1:2,1:npts);
end

% Output some statistics.
fprintf(' Number of integration steps in time = %6d\n', isave(1));
fprintf(' Number of function evaluations      = %6d\n', isave(2));
fprintf(' Number of Jacobian evaluations      = %6d\n', isave(3));
fprintf(' Number of iterations                = %6d\n', isave(5));

% Plot results.
fig1 = figure;
plot_results(x, tsav, squeeze(usav(1,:,:)), 'u_1');

fig2 = figure;
plot_results(x, tsav, squeeze(usav(2,:,:)), 'u_2');

function [p, q, r, ires, user] = pdedef(npde, t, x, nptl, u, ux, ires, user)
  p = zeros(npde, npde, nptl);
  q = zeros(npde, nptl);
  r = zeros(npde, nptl);

  for i = 1:double(nptl)
    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(2,2,i) = 1;
  end;

function [beta, gamma, ires, user] = bndary(npde, t, u, ux, ibnd, ires, user)
  beta  = zeros(npde, 1);
  gamma = zeros(npde, 1);

  beta(1)  = 1;
  beta(2)  = 0;
  gamma(1) = 0;
  if (ibnd == 0)
    gamma(2) = u(1) - 1;
  else
    gamma(2) = u(1) + 1;
  end

function [u, user] = uinit(npde, npts, x, user)
  u = zeros(npde, npts);

  piby2 = pi/2;
  u(1,:) = -sin(piby2*x);
  u(2,:) = -piby2*piby2*u(1,:);

function plot_results(x, t, u, ident)
  % Plot array as a mesh.
  mesh(x, t, u);
  set(gca, 'YScale', 'log');
  set(gca, 'YTick', [0.00001 0.0001 0.001 0.01 0.1 1]);
  set(gca, 'YMinorGrid', 'off');
  set(gca, 'YMinorTick', 'off');

  % Label the axes, and set the title.
  xlabel('x');
  ylabel('t');
  zlabel([ident,'(x,t)']);
  title({['Solution ',ident,' of elliptic-parabolic pair'], ...
      'using Chebyshev Collocation and BDF'});
```

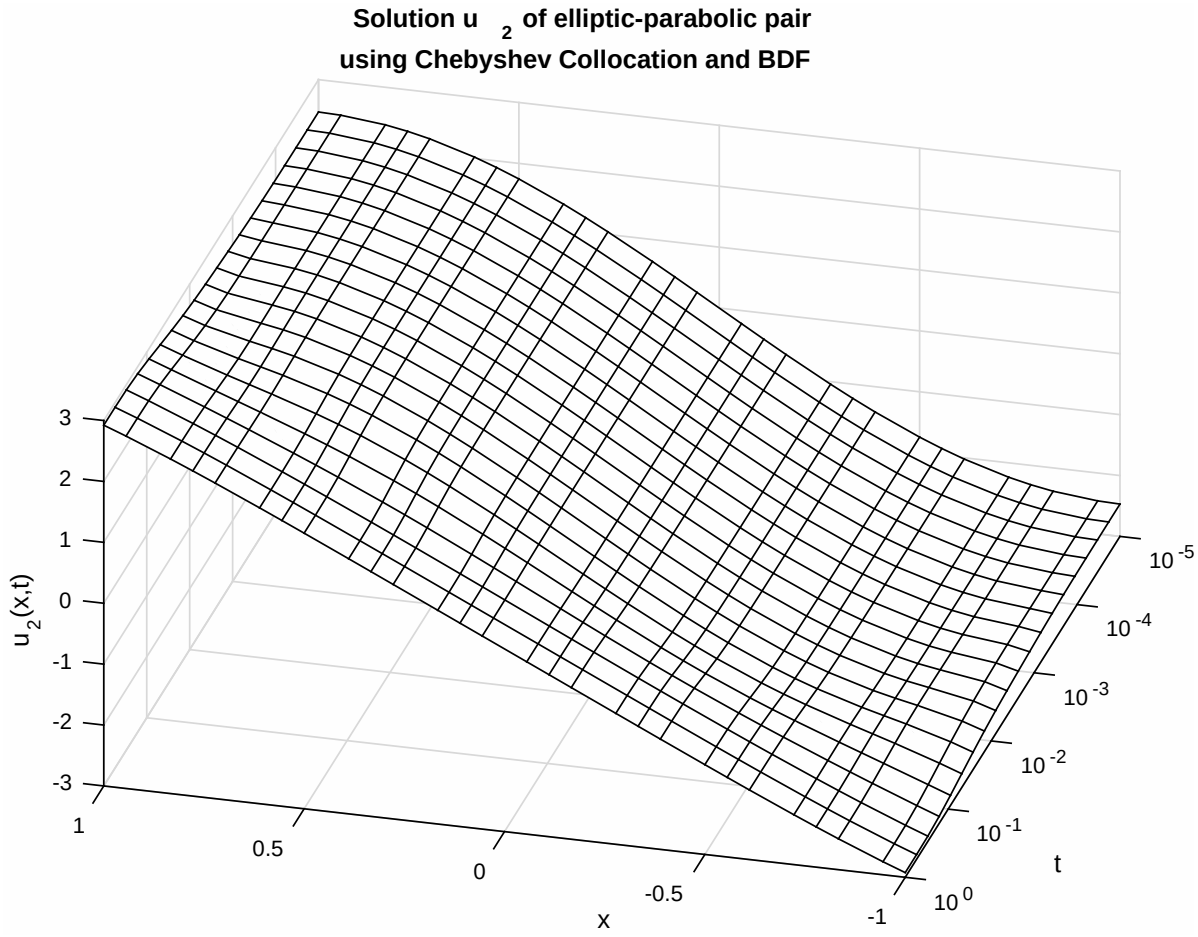
```
  % Set the axes limits tight to the x and y range.
  axis([x(1) x(end) t(1) t(end)]);

  % Set the view to something nice (determined empirically).
  view(-165, 44);
```

## 9.2 Program Results

```
     d03pd example 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

 0.0000  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4690 -1.9961 -0.7624  0.7624  1.9961  2.4690

 0.0000  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4687 -1.9961 -0.7624  0.7624  1.9961  2.4687

 0.0000  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4700 -1.9961 -0.7624  0.7624  1.9961  2.4700

 0.0001  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4724 -1.9960 -0.7624  0.7624  1.9960  2.4724

 0.0001  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4767 -1.9959 -0.7624  0.7624  1.9959  2.4767

 0.0002  u(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
         u(2) -2.4840 -1.9957 -0.7623  0.7623  1.9957  2.4840

 0.0004  u(1)  1.0000  0.8089  0.3090 -0.3090 -0.8089 -1.0000
         u(2) -2.4960 -1.9953 -0.7621  0.7621  1.9953  2.4960

 0.0007  u(1)  1.0000  0.8089  0.3089 -0.3089 -0.8089 -1.0000
         u(2) -2.5142 -1.9946 -0.7619  0.7619  1.9946  2.5142

 0.0013  u(1)  1.0000  0.8087  0.3089 -0.3089 -0.8087 -1.0000
         u(2) -2.5374 -1.9933 -0.7614  0.7614  1.9933  2.5374

 0.0023  u(1)  1.0000  0.8085  0.3087 -0.3087 -0.8085 -1.0000
         u(2) -2.5611 -1.9910 -0.7605  0.7605  1.9910  2.5611

 0.0043  u(1)  1.0000  0.8081  0.3085 -0.3085 -0.8081 -1.0000
         u(2) -2.5869 -1.9866 -0.7588  0.7588  1.9866  2.5869

 0.0078  u(1)  1.0000  0.8074  0.3081 -0.3081 -0.8074 -1.0000
         u(2) -2.6183 -1.9787 -0.7558  0.7558  1.9787  2.6183

 0.0144  u(1)  1.0000  0.8063  0.3075 -0.3075 -0.8063 -1.0000
         u(2) -2.6604 -1.9643 -0.7503  0.7503  1.9643  2.6604

 0.0264  u(1)  1.0000  0.8045  0.3064 -0.3064 -0.8045 -1.0000
         u(2) -2.7128 -1.9394 -0.7402  0.7402  1.9394  2.7128

 0.0483  u(1)  1.0000  0.8020  0.3046 -0.3046 -0.8020 -1.0000
         u(2) -2.7723 -1.9042 -0.7222  0.7222  1.9042  2.7723

 0.0886  u(1)  1.0000  0.7990  0.3022 -0.3022 -0.7990 -1.0000
         u(2) -2.8331 -1.8684 -0.6915  0.6915  1.8684  2.8331

 0.1624  u(1)  1.0000  0.7962  0.2996 -0.2996 -0.7962 -1.0000
         u(2) -2.8840 -1.8423 -0.6512  0.6512  1.8423  2.8840

 0.2976  u(1)  1.0000  0.7944  0.2978 -0.2978 -0.7944 -1.0000
         u(2) -2.9140 -1.8287 -0.6218  0.6218  1.8287  2.9140

 0.5456  u(1)  1.0000  0.7939  0.2973 -0.2973 -0.7939 -1.0000
```

```
         u(2) -2.9225 -1.8250 -0.6129  0.6129  1.8250  2.9225

 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 =     38
 Number of function evaluations      =    237
 Number of Jacobian evaluations      =      9
 Number of iterations                =     89
```




Solution u 2 of elliptic-parabolic pair
using Chebyshev Collocation and BDF
u2(x,t)
3
2
1
0
-1
-2
-3
1
0.5
0
-0.5
-1
x
10 -5
10 -4
10 -3
10 -2
10 -1
10 0
t