NAG Library Function Document

nag_pde_parab_1d_fd (d03pcc)

1 Purpose

nag_pde_parab_1d_fd (d03pcc) integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable. The spatial discretization is performed using finite differences, 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_fd (Integer npde, Integer m, double *ts, double tout,
     void  (*pdedef)(Integer npde, double t, double x, 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 npts, const double x[], 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_fd (d03pcc) integrates the system of parabolic equations:

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

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 $\frac{\partial U}{\partial t}$.

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 $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$. 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 mesh should be chosen in accordance with the expected behaviour of the solution.

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 $t = t_0$. 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**.

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 the mid-points of the mesh intervals by calling the **pdedef** for each mid-point in turn. Any discontinuities in these functions **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathbf{npts}}$;

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

(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 mesh points. For simple problems in Cartesian coordinates, this system is obtained by replacing the space derivatives by the usual central, three-point finite difference formula. However, for polar and spherical problems, or problems with nonlinear coefficients, the space derivatives are replaced by a modified three-point formula which maintains second-order accuracy. In total there are **npde** $\times$ **npts** ODEs in the time direction. This 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, 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

Dew P M and Walsh J (1981) A set of library routines for solving parabolic equations in one space variable *ACM Trans. Math. Software* **7** 295–314

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

5 Arguments

1: **npde** – Integer *Input*

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

Constraint: **npde** ≥ 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 functions $P_{i,j}$, Q_i and R_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_fd (d03pcc).

The specification of **pdedef** is:

```
void  pdedef (Integer npde, double t, double x, 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** – double *Input*

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

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

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

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

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

6: **p**[**npde** × **npde**] – double *Output*

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

7: **q**[**npde**] – double *Output*

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

8: **r**[**npde**] – double *Output*

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

9: **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_fd (d03pcc) returns to the calling function with the error indicator set to **fail**.**code** = NE_FAILED_DERIV.

10: **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_fd (d03pcc) you may allocate memory and initialize these pointers with various quantities for use by **pdedef** when called from nag_pde_parab_1d_fd (d03pcc) (see Section 3.2.1.1 in the Essential Introduction).

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 $\frac{\partial U_i(x,t)}{\partial x}$ at the boundary specified by **ibnd**, for $i = 1, 2, \ldots,$ **npde**.

5: **ibnd** – Integer *Input*

On entry: determines the position of the boundary conditions.

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

ibnd $\neq$ 0
Indicates that **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_fd (d03pcc) returns to the calling function with the error indicator set to **fail.code** = NE_FAILED_DERIV.

9: **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_fd (d03pcc) you may allocate memory and initialize these pointers with various quantities for use by **bndary** when called from nag_pde_parab_1d_fd (d03pcc) (see Section 3.2.1.1 in the Essential Introduction).

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

On entry: the initial values of $U(x,t)$ at $t=$ **ts** and the mesh points **x**$[j-1]$, for $j=1,2,\ldots,$**npts**.

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

8: **npts** – Integer *Input*

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

Constraint: **npts** ≥ 3.

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

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

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

10: **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: **acc** > 0.0.

11: **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.

12: **lrsave** – Integer *Input*

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

Constraint: **lrsave** $\geq (6 \times \mathbf{npde} + 10) \times \mathbf{npde} \times \mathbf{npts} + (3 \times \mathbf{npde} + 21) \times \mathbf{npde} + 7 \times \mathbf{npts} + 54$.

13: **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.

14: **lisave** – Integer *Input*

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

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

15: **itask** – Integer *Input*

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

itask $= 1$
Normal computation of output values **u** at $t = \mathbf{tout}$.

itask $= 2$
One step and return.

itask $= 3$
Stop at first internal integration point at or beyond $t = \mathbf{tout}$.

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

16: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_pde_parab_1d_fd (d03pcc) 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.

17: **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.

18: **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_fd (d03pcc).

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

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

19: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

20: **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.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

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.

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

acc was too small to start integration: **acc** $=$ ⟨*value*⟩.

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 value of **acc**. **ts** $=$ ⟨*value*⟩, **acc** $=$ ⟨*value*⟩.

NE_INCOMPAT_PARAM

On entry, **m** $=$ ⟨*value*⟩ and **x**$[0] =$ ⟨*value*⟩.
Constraint: **m** ≤ 0 or **x**$[0] \geq 0.0$

NE_INT

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

On entry, **ind** $=$ ⟨*value*⟩.
Constraint: **ind** $= 0$ or 1.

On entry, **itask** $=$ ⟨*value*⟩.
Constraint: **itask** $= 1$, 2 or 3.

On entry, **m** $=$ ⟨*value*⟩.
Constraint: **m** $= 0$, 1 or 2.

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

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

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_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.

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

NE_NOT_CLOSE_FILE

Cannot close file ⟨*value*⟩.

NE_NOT_STRICTLY_INCREASING

On entry, mesh points **x** appear to be badly ordered: $I =$ ⟨*value*⟩, **x**$[I-1] =$ ⟨*value*⟩, $J =$ ⟨*value*⟩ and **x**$[J-1] =$ ⟨*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_fd (d03pcc) 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 argument, **acc**.

8 Parallelism and Performance

nag_pde_parab_1d_fd (d03pcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_pde_parab_1d_fd (d03pcc) 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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

nag_pde_parab_1d_fd (d03pcc) is designed to solve parabolic systems (possibly including some 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. It may be advisable in such cases to reduce the whole system to first-order and to use the Keller box scheme function nag_pde_parab_1d_keller (d03pec).

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

10 Example

We use the example given in Dew and Walsh (1981) which consists of an elliptic-parabolic pair of PDEs. The problem was originally derived from a single third-order in space PDE. The elliptic equation is

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r^2\frac{\partial U_1}{\partial r}\right) = 4\alpha\left(U_2 + r\frac{\partial U_2}{\partial r}\right)$$

and the parabolic equation is

$$(1-r^2)\frac{\partial U_2}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\left(\frac{\partial U_2}{\partial r} - U_2U_1\right)\right)$$

where $(r,t) \in [0,1] \times [0,1]$. The boundary conditions are given by

$$U_1 = \frac{\partial U_2}{\partial r} = 0 \quad \text{at } r = 0,$$

and

$$\frac{\partial}{\partial r}(rU_1) = 0 \quad \text{and} \quad U_2 = 0 \quad \text{at } r = 1.$$

The first of these boundary conditions implies that the flux term in the second PDE, $\left(\frac{\partial U_2}{\partial r} - U_2U_1\right)$, is zero at $r = 0$.

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

$$U_1 = 2\alpha r \quad \text{and} \quad U_2 = 1.0, \qquad r \in [0,1].$$

The value $\alpha = 1$ was used in the problem definition. A mesh of 20 points was used with a circular mesh spacing to cluster the points towards the right-hand side of the spatial interval, $r = 1$.

10.1 Program Text

```
/* nag_pde_parab_1d_fd (d03pcc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 * Mark 23 revised, 2011.
 */

#include <stdio.h>
#include <string.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 pdedef(Integer, double, double, 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 *);
static int NAG_CALL uinit(double *, double *, Integer, Integer, double);
#ifdef __cplusplus
}
#endif
```

```
int main(void)
{
  const Integer npts = 20, npde = 2, neqn = npts*npde, intpts = 6, itype = 1;
  const Integer nwk = (10+6*npde)*neqn, lisave = neqn+24;
  const Integer lrsave = nwk+(21+3*npde)*npde+7*npts+54;
  static double ruser[2] = {-1.0, -1.0};
  Integer       exit_status = 0, i, ind, it, itask, itrace, m;
  double        acc, alpha, hx, piby2, tout, ts;
  double        xout[6] = { 0., .4, .6, .8, .9, 1. };
  double        *rsave = 0, *u = 0, *uout = 0, *x = 0;
  Integer       *isave = 0;
  NagError      fail;
  Nag_Comm      comm;
  Nag_D03_Save  saved;

  INIT_FAIL(fail);

  printf("nag_pde_parab_1d_fd (d03pcc) 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)) ||
      !(isave = NAG_ALLOC(lisave, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = 1;
      goto END;
    }

  acc = 0.001;
  m = 1;
  itrace = 0;
  alpha = 1.0;
  comm.p = (Pointer)
  ind = 0;
  itask = 1;

  /* Set spatial mesh points */

  piby2 = 0.5*nag_pi;
  hx = piby2/((double)(npts-1));
  x[0] = 0.0;
  x[npts-1] = 1.0;
  for (i = 1; i < npts-1; ++i)  x[i] = sin(hx*i);

  /* Set initial conditions */

  ts = 0.0;
  tout = 1e-5;

  printf("Accuracy requirement  = %12.5f\n", acc);
  printf("Parameter alpha       = %10.3f\n\n", alpha);
  printf("   t  /  x  ");

  for (i = 0; i < intpts; ++i) printf("%8.4f", xout[i]);
  printf("\n");

  /* Set the initial values */

  uinit(u, x, npde, npts, alpha);
  for (it = 0; it < 5; ++it)
    {
      tout *= 10.0;

      /* Solve for next iteration step using
       * nag_pde_parab_1d_fd (d03pcc).
```

```
       * General system of parabolic PDEs, method of lines, finite
       * differences, one space variable
       */
      nag_pde_parab_1d_fd(npde, m, &ts, tout, pdedef, bndary, u, npts, x, acc,
                          rsave, lrsave, isave, lisave, itask, itrace, 0, &ind,
                          &comm, &saved, &fail);

      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_pde_parab_1d_fd (d03pcc).\n%s\n",
                 fail.message);
          exit_status = 1;
          goto END;
        }

      /* Interpolate at required spatial points using
       * nag_pde_interp_1d_fd (d03pzc).
       * PDEs, spatial interpolation fo use with the suite of routines
       * nag_pde_parab_1d (d03p).
       */
      nag_pde_interp_1d_fd(npde, m, u, npts, x, xout, intpts, 1, uout, &fail);

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

      printf("\n %6.4f u(1)", tout);
      for (i = 0; i < intpts; ++i) printf("%8.4f", uout[npde*i]);

      printf("\n %6s u(2)","");
      for (i = 0; i < intpts; ++i) printf("%8.4f", uout[npde*i+1]);
      printf("\n");
    }

  /* Print integration statistics */

  printf("\n %-55s%4ld\n", "Number of integration steps in time",
         isave[0]);
  printf(" %-55s%4ld\n", "Number of residual evaluations of"
         " resulting ODE system", isave[1]);
  printf(" %-55s%4ld\n", "Number of Jacobian evaluations", isave[2]);
  printf(" %-55s%4ld\n", "Number of iterations of nonlinear solver",
         isave[4]);

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

  return exit_status;
}

static int NAG_CALL uinit(double *u, double *x, Integer npde, Integer npts,
                          double alpha)
{
  Integer i;

  /* Intial conditions for u1 */
  for (i = 0; i < npts; ++i)  u[i*npde] = alpha*2.0*x[i];
  /* Intial conditions for u2 */
  for (i = 0; i < npts; ++i)  u[i*npde+1] = 1.0;

  return 0;
}
```

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

  double *alpha = (double *) comm->p;

  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback pdedef, first invocation.)\n");
      comm->user[0] = 0.0;
    }
  /* Coefficients on first PDE */
  q[0] = *alpha*4.0*(u[1]+x*ux[1]);
  r[0] = x*ux[0];
  p[0] = 0.0;
  p[npde] = 0.0;
  /* Coefficients on first PDE */
  q[1] = 0.0;
  r[1] = ux[1] - u[0] * u[1];
  p[1] = 0.0;
  p[1+npde] = 1.0-x*x;
  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)
{
  /* Boundary conditions */

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback bndary, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  if (ibnd == 0)
    {
      /* u[0] = 0 */
      beta[0] = 0.0;
      gamma[0] = u[0];
      /* ux[1] = 0 ==> 1.0*r[1] = ux[1] - u[0]*u[1] = -u[0]*u[1] */
      beta[1] = 1.0;
      gamma[1] = -u[0]*u[1];
    }
  else
    {
      /* d(x*u[0])/dx = x*ux[0] + u[0] = 0 */
      beta[0] = 1.0;
      gamma[0] = -u[0];
      /* u[1] = 0 */
      beta[1] = 0.0;
      gamma[1] = u[1];
    }
  return;
}
```

10.2 Program Data

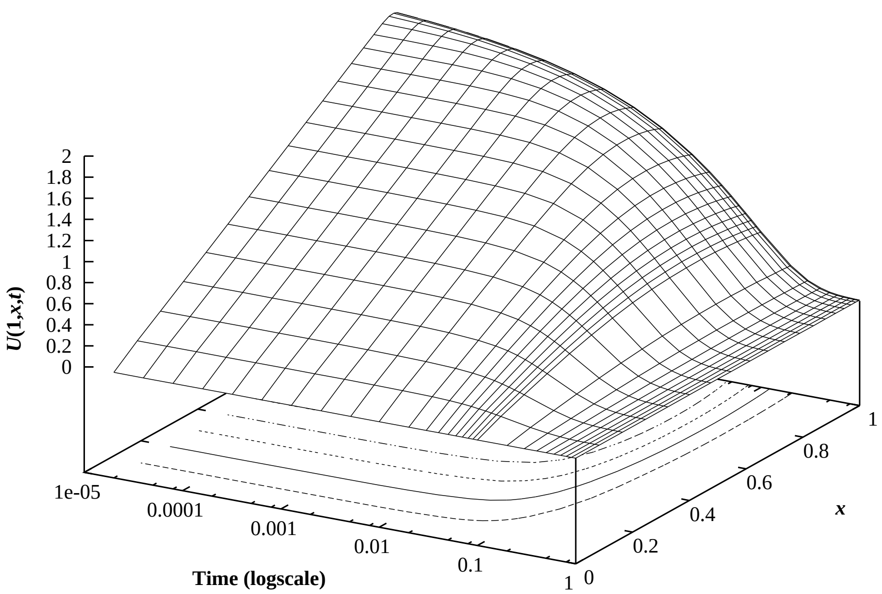
None.

10.3 Program Results

```
nag_pde_parab_1d_fd (d03pcc) Example Program Results

Accuracy requirement  =      0.00100
Parameter alpha       =      1.000
```

```
   t  /  x    0.0000  0.4000  0.6000  0.8000  0.9000  1.0000
(User-supplied callback bndary, first invocation.)
(User-supplied callback pdedef, first invocation.)

 0.0001 u(1)  0.0000  0.8008  1.1988  1.5990  1.7958  1.8485
        u(2)  0.9997  0.9995  0.9994  0.9988  0.9663 -0.0000

 0.0010 u(1)  0.0000  0.7982  1.1940  1.5841  1.7179  1.6734
        u(2)  0.9969  0.9952  0.9937  0.9484  0.6385 -0.0000

 0.0100 u(1)  0.0000  0.7676  1.1239  1.3547  1.3635  1.2830
        u(2)  0.9627  0.9495  0.8754  0.5537  0.2908 -0.0000

 0.1000 u(1)  0.0000  0.3908  0.5007  0.5297  0.5120  0.4744
        u(2)  0.5468  0.4299  0.2995  0.1479  0.0724 -0.0000

 1.0000 u(1)  0.0000  0.0007  0.0008  0.0008  0.0008  0.0007
        u(2)  0.0010  0.0007  0.0005  0.0002  0.0001 -0.0000

 Number of integration steps in time                      78
 Number of residual evaluations of resulting ODE system  378
 Number of Jacobian evaluations                           25
 Number of iterations of nonlinear solver                190
```

Example Program
Solution, $U(1,x,t)$, of Elliptic-parabolic Pair using Method of Lines and BDF Method



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