NAG Toolbox

nag_pde_1d_parab_fd (d03pc)

1 Purpose

nag_pde_1d_parab_fd (d03pc) 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 Syntax

```
[ts, u, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] =
nag_pde_1d_parab_fd(m, ts, tout, pdedef, bndary, u, x, acc, rsave, isave, itask,
itrace, ind, cwsav, lwsav, iwsav, rwsav, 'npde', npde, 'npts', npts, 'user',
user)
[ts, u, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] = d03pc(m,
ts, tout, pdedef, bndary, u, x, acc, rsave, isave, itask, itrace, ind, cwsav,
lwsav, iwsav, rwsav, 'npde', npde, '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_fd (d03pc) integrates the system of parabolic equations:

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

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_{out}, over the space interval $a \le x \le b$, where $a = x_1$ and $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a 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 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 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_1d_parab_fd (d03pc).

```
[p, q, r, ires, user] = pdedef(npde, t, x, 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** – REAL (KIND=nag_wp)

The current value of the space variable x.

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

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

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

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

6: **ires** – INTEGER

Set to -1 or 1.

7: **user** – INTEGER array

pdedef is called from nag_pde_1d_parab_fd (d03pc) with the object supplied to nag_pde_1d_parab_fd (d03pc).

Output Parameters

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

p(i,j) 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**.

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

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

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

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

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_fd (d03pc) 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 β_i and γ_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

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: **ires** – INTEGER

Set to -1 or 1.

7: **user** – INTEGER array

bndary is called from nag_pde_1d_parab_fd (d03pc) with the object supplied to nag_pde_1d_parab_fd (d03pc).

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_fd (d03pc) 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

The initial values of $U(x,t)$ at $t =$ **ts** and the mesh points **x**(j), for $j = 1,2,\ldots,$ **npts**.

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

The mesh points in the spatial direction. **x**(1) must specify the left-hand boundary, a, and **x**(**npts**) must specify the right-hand boundary, b.

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

8: **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 jth mesh point, the error test is:

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

Constraint: **acc** > 0.0.

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

$lrsave$, the dimension of the array, must satisfy the constraint $lrsave \geq (6 \times \textbf{npde} + 10) \times \textbf{npde} \times \textbf{npts} + (3 \times \textbf{npde} + 21) \times \textbf{npde} + 7 \times \textbf{npts} + 54$.

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.

10: **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.

11: **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.

12: **itrace** – INTEGER

The level of trace information required from nag_pde_1d_parab_fd (d03pc) 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.

13: **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_fd (d03pc).

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

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

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

If **ind** = 1, **cwsav** must be unchanged from the previous call to the function.

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

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

If **ind** = 1, **lwsav** must be unchanged from the previous call to the function.

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

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

If **ind** = 1, **iwsav** must be unchanged from the previous call to the function.

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

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

If **ind** = 1, **rwsav** must be unchanged from the previous call to the function.

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** ≥ 1.

2: **npts** – INTEGER

Default: the dimension of the array **x** and the second dimension of the array **u**. (An error is raised if these dimensions are not equal.)

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

Constraint: **npts** ≥ 3.

3: **user** – INTEGER array

user is not used by nag_pde_1d_parab_fd (d03pc), but is passed to **pdedef** and **bndary**. 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: **rsave**($lrsave$) – REAL (KIND=nag_wp) array

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

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

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

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.

5: **ind** – INTEGER

ind $= 1$.

6: **user** – INTEGER array

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

If **ind** $= 1$, **cwsav** must be unchanged from the previous call to the function.

8: **lwsav**(**100**) – LOGICAL array

If **ind** $= 1$, **lwsav** must be unchanged from the previous call to the function.

9: **iwsav**(**505**) – INTEGER array

If **ind** $= 1$, **iwsav** must be unchanged from the previous call to the function.

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

If **ind** $= 1$, **rwsav** must be unchanged from the previous call to the function.

11: **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 **x**$(1) < 0.0$,
or the mesh points **x**(i) are not ordered,
or **npts** < 3,
or **npde** < 1,

or **acc** ≤ 0.0,
or **ind** $\neq 0$ or 1,
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_fd (d03pc) 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 Further Comments

nag_pde_1d_parab_fd (d03pc) 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_1d_parab_keller (d03pe).

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

9 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

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

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, $r = 1$.

9.1 Program Text

```
     function  d03pc_example

fprintf('d03pc example results\n\n');

%  Solution of an elliptic-parabolic pair of PDEs.
global alpha;

% Set values for problem parameters.
npde = 2;

% Number of points on calculation mesh, and on interpolated mesh.
npts = 20;
intpts = 6;

itype  = nag_int(1);
neqn   = npde*npts;
lisave = neqn + 24;
nwk    = (10 + 6*npde)*neqn;
lrsave = nwk + (21 + 3*npde)*npde + 7*npts + 54;

% Define some arrays.
rsave  = zeros(lrsave, 1);
u      = zeros(npde, npts);
uinterp = zeros(npde, intpts, itype);
x     = zeros(npts, 1);
isave = zeros(lisave, 1, nag_int_name);
cwsav = {'', '', '', '', '', '', '', '', '', ''};
lwsav = false(100, 1);
iwsav = zeros(505, 1, nag_int_name);
rwsav = zeros(1100, 1);

% Set up the points on the interpolation grid.
xinterp = [0.0 0.4 0.6 0.8 0.9 1.0];

acc = 1.0e-3;
alpha = 1;

itrace = nag_int(0);
itask  = nag_int(1);
% Use cylindrical polar coordinates.
m = nag_int(1);

% We run through the calculation twice; once to output the interpolated
% results, and once to store the results for plotting.
niter = [5, 28];

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

for icalc = 1:2

  % Set spatial mesh points.
  piby2 = pi/2;
  hx = piby2/(npts-1);
  x = sin(0:hx:piby2);

  % Set initial conditions.
  ts   = 0;
  tout = 1e-5;
```

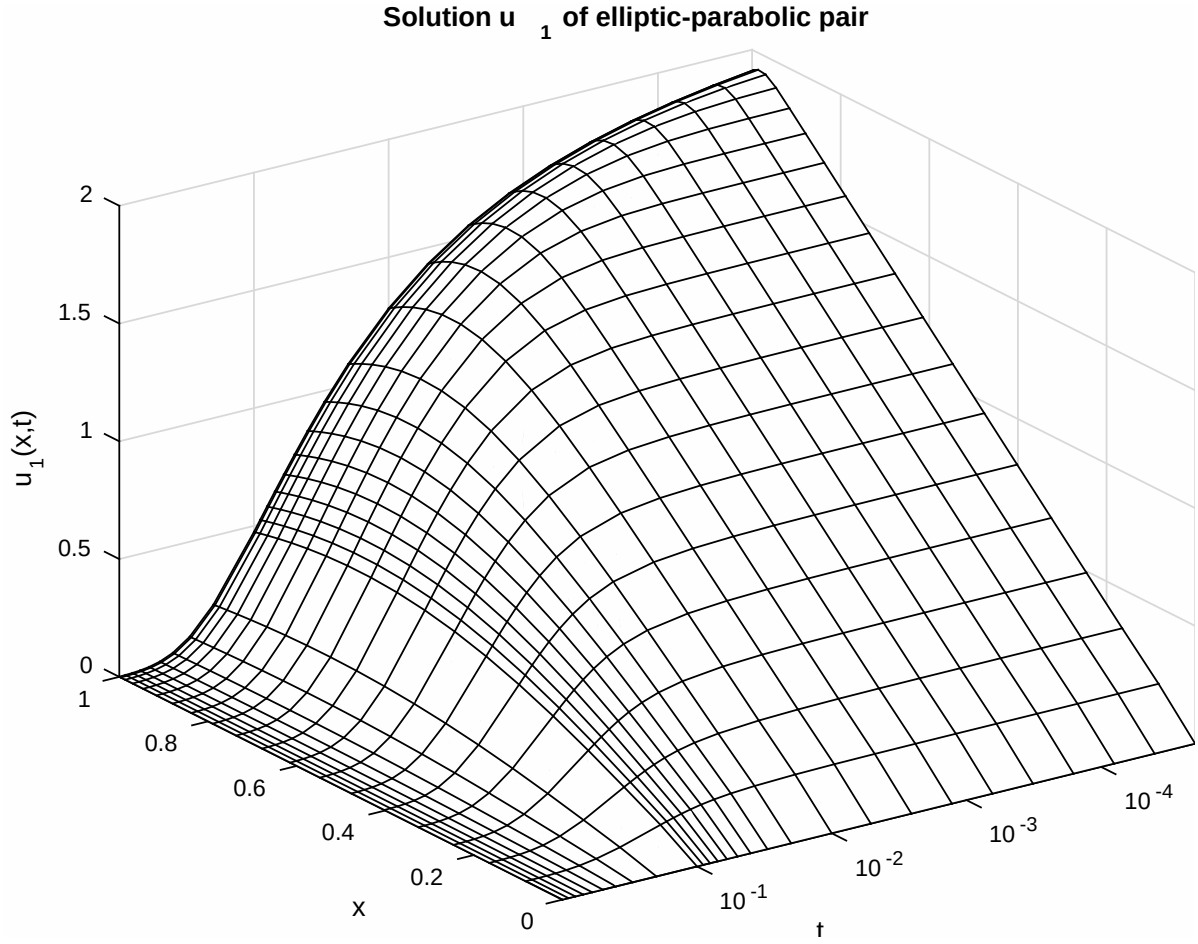
```
% Set the initial values.
[u] = uinit(x, npts);

% Start the integration in time.
ind = nag_int(0);

% Counter for saved results.
isav = 0;

% Loop over endpoints for the integration.
% Set itask = 1: normal computation of output values at t = tout.
for iter = 1:niter(icalc)

  %Set the endpoint.
  if icalc == 1
    tout = 10.0*tout;
  else
    if iter < 10
      tout = 2*tout;
    else
      if iter == 10
        tout = 0.01;
      else
        if iter < 20
          tout = tout + 0.01;
        else
          tout = tout + 0.1;
        end
      end
    end
  end

  % The first time this is called, ind is 0, which (re)starts the
  % integration in time.  On exit, ind is set to 1; using this value
  % on a subsequent call continues the integration.  This means that
  % only tout and ifail should be reset between calls.
  [ts, u, rsave, isave, ind, user, cwsav, lwsav, iwsav, rwsav, ifail] = ...
  d03pc(...
         m, ts, tout, @pdedef, @bndary, u, x, acc, rsave, isave, itask, ...
        itrace, ind, cwsav, lwsav, iwsav, rwsav);

  if icalc == 1
    % Output interpolation points first time through.
    if iter == 1
      fprintf([' accuracy requirement = %12.5e\n', ...
               ' parameter alpha =  %12.3e\n'], acc, alpha);
      fprintf(' t / x       ');
      fprintf('%8.4f', xinterp(1:intpts));
      fprintf('\n\n');
    end

    % Call d03pz to do interpolation of results onto coarser grid.
    [uinterp, ifail] = d03pz( ...
                              m, u, x, xinterp, itype);

    % Output interpolated results for this time step.
    fprintf('%7.4f  u(1)', tout);
    fprintf('%8.4f', uinterp(1,1:intpts,1));
    fprintf('\n%13s','u(2)');
    fprintf('%8.4f', uinterp(2,1:intpts,1));
    fprintf('\n\n');
  else
    % Save this timestep, and this set of results.
    isav = isav+1;
    tsav(isav) = ts;
    usav(1:2,isav,1:npts) = u(1:2,1:npts);
  end
end

if icalc == 1
```

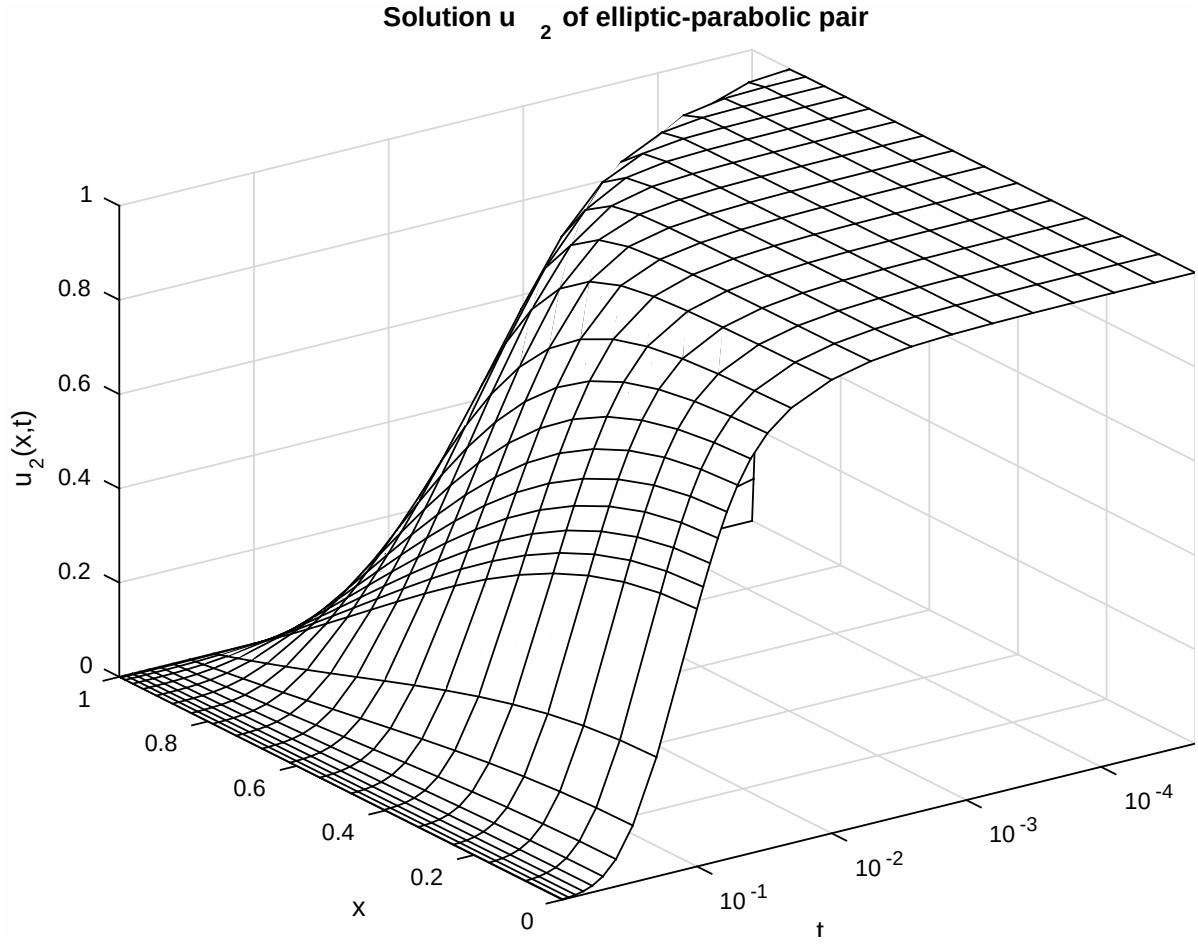
```
    % Output some statistics.
    fprintf([' Number of integration steps in time = %6d\n', ...
             ' Number of function evaluations      = %6d\n', ...
             ' Number of Jacobian evaluations      = %6d\n', ...
             ' Number of iterations                = %6d\n'], ...
            isave(1), isave(2), isave(3), isave(5));
  else
    % Plot results.
    fig1 = figure;
    plot_results(x, tsav, squeeze(usav(1,:,:)), 'u_1');
    % print(fig1,'-dpng','-r75','d03pc_fig1.png');
    % print(fig1,'-deps','-r75','d03pc_fig1.eps');
    fig2 = figure;
    plot_results(x, tsav, squeeze(usav(2,:,:)), 'u_2');
    % print(fig2,'-dpng','-r75','d03pc_fig2.png');
    % print(fig2,'-deps','-r75','d03pc_fig2.eps');
  end
end

function [p, q, r, ires, user] = pdedef(npde, t, x, u, ux, ires, user)
  % Evaluate Pij, Qi and Ri which define the system of PDEs.
  global alpha;

  p = zeros(npde,npde);
  q = zeros(npde,1);
  r = zeros(npde,1);
  q(1) = 4*alpha*(u(2) + x*ux(2));
  r(1) = x*ux(1);
  r(2) = ux(2) - u(1)*u(2);
  p(2,2) = 1 - x*x;

function [beta, gamma, ires, user] = bndary(npde, t, u, ux, ibnd, ires, user)
  % Evaluate beta and gamma to define the boundary conditions.

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

function [u] = uinit(x, npts)
  % Set initial values for solution.
  global alpha;

  for i = 1:npts
    u(1,i) = 2*alpha*x(i);
    u(2,i) = 1;
  end

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 method of lines and BDF'});
  title(['Solution ',ident,' of elliptic-parabolic pair']);
```

```
% 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(-125, 30);
```

9.2 Program Results

```
    d03pc example results

accuracy requirement =  1.00000e-03
parameter alpha =     1.000e+00
t / x         0.0000  0.4000  0.6000  0.8000  0.9000  1.0000

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 function evaluations      =    378
Number of Jacobian evaluations      =     25
Number of iterations                =    190
```


Solution u 1 of elliptic-parabolic pair
u 1(x,t)
2
1.5
1
0.5
0
1
0.8
0.6
0.4
0.2
0
x
10 -1
10 -2
10 -3
10 -4
t


Solution u 2 of elliptic-parabolic pair
u2(x,t)
1
0.8
0.6
0.4
0.2
0
1
0.8
0.6
0.4
0.2
0
x
10 -1
10 -2
10 -3
10 -4
t