NAG Toolbox for MATLAB

d03pe

1 Purpose

d03pe integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable. The spatial discretization is performed using the Keller box scheme 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 (BDF) method.

2 Syntax

```
[ts, u, rsave, isave, ind, ifail] = d03pe(ts, tout, pdedef, bndary, u,
x, nleft, acc, rsave, isave, itask, itrace, ind, 'npde', npde, 'npts',
npts, 'lrsave', lrsave, 'lisave', lisave)
```

3 Description

d03pe integrates the system of first-order PDEs

$$G_i(x,t,U,U_x,U_t) = 0, \qquad i = 1,2,\ldots,\mathbf{npde}. \tag{1}$$

In particular the functions G_i must have the general form

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j}\frac{\partial U_j}{\partial t} + Q_i, \qquad i = 1,2,\ldots,\mathbf{npde}, \qquad a \le x \le b, t \ge t_0, \tag{2}$$

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

$$U(x,t) = \left[U_1(x,t),\ldots,U_{\mathbf{npde}}(x,t)\right]^{\mathrm{T}}, \tag{3}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$ and Q_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 mesh should be chosen in accordance with the expected behaviour of the solution.

The PDE system which is defined by the functions G_i must be specified in a user-supplied (sub)program **pdedef**.

The initial values of the functions $U(x,t)$ must be given at $t = t_0$. For a first-order system of PDEs, only one boundary condition is required for each PDE component U_i. The **npde** boundary conditions are separated into n_a at the left-hand boundary $x = a$, and n_b at the right-hand boundary $x = b$, such that $n_a + n_b = \mathbf{npde}$. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of U_i at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for U_i should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialization or integration difficulties in the underlying time integration functions.

The boundary conditions have the form:

$$G_i^L(x,t,U,U_t) = 0 \qquad \text{at } x = a, \qquad i = 1,2,\ldots,n_a \tag{4}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,U_t) = 0 \qquad \text{at } x = b, \qquad i = 1,2,\ldots,n_b \tag{5}$$

at the right-hand boundary.

Note that the functions G_i^L and G_i^R must not depend on U_x, since spatial derivatives are not determined explicitly in the Keller box scheme (see Keller 1970). If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that G_i^L and G_i^R must be linear with respect to time derivatives, so that the boundary conditions have the general form

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L = 0, \qquad i = 1, 2, \ldots, n_a \tag{6}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathbf{npde}} E_{i,j}^R \frac{\partial U_j}{\partial t} + S_i^R = 0, \qquad i = 1, 2, \ldots, n_b \tag{7}$$

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

The boundary conditions must be specified in a user-supplied (sub)program **bndary**.

The problem is subject to the following restrictions:

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

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

(iii) The evaluation of the function G_i is done at the mid-points of the mesh intervals by calling the user-supplied (sub)program **pdedef** for each mid-point in turn. Any discontinuities in the function **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.

In this method of lines approach the Keller box scheme (see Keller 1970) is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are **npde** $\times$ **npts** ODEs in the time direction. This system is then integrated forwards in time using a BDF method.

4 References

Berzins M 1990 Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (ed J C Mason and M G Cox) 59–72 Chapman and Hall

Berzins M, Dew P M and Furzeland R M 1989 Developing software for time-dependent problems using the method of lines and differential-algebraic integrators *Appl. Numer. Math.* **5** 375–397

Keller H B 1970 A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** 327–350 Academic Press

Pennington S V and Berzins M 1994 New NAG Library software for first-order partial differential equations *ACM Trans. Math. Softw.* **20** 63–99

5 Parameters

5.1 Compulsory Input Parameters

1: **ts – double scalar**

The initial value of the independent variable t.

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

2: **tout – double scalar**

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

3: **pdedef – string containing name of m-file**

pdedef must compute the functions G_i which define the system of PDEs. **pdedef** is called approximately midway between each pair of mesh points in turn by d03pe.

Its specification is:

```
[res, ires] = pdedef(npde, t, x, u, ut, ux, ires)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **x – double scalar**

The current value of the space variable x.

4: **u(npde) – double array**

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

5: **ut(npde) – double array**

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

6: **ux(npde) – double array**

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

7: **ires – int32 scalar**

The form of G_i that must be returned in the array **res**.

ires $= -1$

Equation (8) must be used.

ires $= 1$

Equation (9) must be used.

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)program 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 d03pe returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **res**(**npde**) – **double array**

res(i) must contain the ith component of G, for $i = 1, 2, \ldots,$ **npde**, where G is defined as

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j} \frac{\partial U_j}{\partial t}, \tag{8}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i = \sum_{j=1}^{\mathbf{npde}} P_{i,j} \frac{\partial U_j}{\partial t} + Q_i, \tag{9}$$

i.e., all terms in equation (2).

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

2: **ires – int32 scalar**

The form of G_i that must be returned in the array **res**.

ires $= -1$

Equation (8) must be used.

ires $= 1$

Equation (9) must be used.

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)program 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 d03pe returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

4: **bndary – string containing name of m-file**

bndary must compute the functions G_i^L and G_i^R which define the boundary conditions as in equations (4) and (5).

Its specification is:

```
[res, ires] = bndary(npde, t, ibnd, nobc, u, ut, ires)
```

Input Parameters

1: **npde – int32 scalar**

The number of PDEs in the system.

2: **t – double scalar**

The current value of the independent variable t.

3: **ibnd – int32 scalar**

Determines the position of the boundary conditions.

ibnd $= 0$

bndary must compute the left-hand boundary condition at $x = a$.

ibnd $\neq 0$

Indicates that **bndary** must compute the right-hand boundary condition at $x = b$.

4: **nobc – int32 scalar**

Specifies the number of boundary conditions at the boundary specified by **ibnd**.

5: **u**(**npde**) **– double 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**.

6: **ut**(**npde**) **– double array**

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

7: **ires – int32 scalar**

The form G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$

Equation (10) must be used.

ires $= 1$

Equation (11) must be used.

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)program 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 d03pe returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

Output Parameters

1: **res**(**nobc**) **– double array**

res(i) must contain the ith component of G^L or G^R, depending on the value of **ibnd**, for $i = 1, 2, \ldots,$ **nobc**, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t}, \tag{10}$$

i.e., only terms depending explicitly on time derivatives, or

$$G_i^L = \sum_{j=1}^{\mathbf{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L, \tag{11}$$

i.e., all terms in equation (6), and similarly for G_i^R.
The definitions of G^L and G^R are determined by the input value of **ires**.

2: **ires – int32 scalar**

The form G_i^L (or G_i^R) that must be returned in the array **res**.

ires $= -1$

Equation (10) must be used.

ires $= 1$

Equation (11) must be used.

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)program 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 d03pe returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

5: **u(npde,npts) – double array**

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

6: **x(npts) – double 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**).

7: **nleft – int32 scalar**

The number n_a of boundary conditions at the left-hand mesh point **x**(1).

Constraint: $0 \leq$ **nleft** $\leq$ **npde**.

8: **acc – double scalar**

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:

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

Constraint: **acc** > 0.0.

9: **rsave(lrsave) – double 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.

10: **isave**(**lisave**) – **int32 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(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 – int32 scalar**

Specifies the task to be performed by the ODE integrator.

itask = 1

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

itask = 2

Take one step and return.

itask = 3

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

Constraint: $1 \leq$ **itask** ≤ 3.

12: **itrace – int32 scalar**

The level of trace information required from d03pe and the underlying ODE solver as follows:

itrace ≤ -1

No output is generated.

itrace $= 0$

Only warning messages from the PDE solver are printed on the current error message unit (see x04aa).

itrace $= 1$

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

itrace $= 2$

Output from the underlying ODE solver is similar to that produced when **itrace** $= 1$, except that the advisory messages are given in greater detail.

itrace ≥ 3

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

You are advised to set **itrace** $= 0$, unless you are experienced with sub-chapter D02M/N.

13: **ind – int32 scalar**

Must be set to 0 or 1.

ind $= 0$

Starts or restarts the integration in time.

ind $= 1$

Continues the integration after an earlier exit from the function. In this case, only the parameters **tout** and **ifail** should be reset between calls to d03pe.

Constraint: $0 \leq$ **ind** ≤ 1.

5.2 Optional Input Parameters

1: **npde – int32 scalar**

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

the number of PDEs in the system to be solved.

Constraint: **npde** ≥ 1.

2: **npts – int32 scalar**

Default: The dimension of the arrays **u**, **x**. (An error is raised if these dimensions are not equal.)

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

Constraint: **npts** ≥ 3.

3: **lrsave – int32 scalar**

Default: The dimension of the array **rsave**.

Constraint: **lrsave** $\geq (4 \times$ **npde** $+$ **nleft** $+ 14) \times$ **npde** $\times$ **npts** $+ (3 \times$ **npde** $+ 21) \times$ **npde** $+$ $7 \times$ **npts** $+ 54$.

4: **lisave – int32 scalar**

Default: The dimension of the array **isave**.

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

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **ts – double scalar**

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

2: **u**(**npde**,**npts**) **– double array**

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

3: **rsave**(**lrsave**) **– double 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.

4: **isave**(**lisave**) **– int32 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(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 – int32 scalar**

ind $= 1$.

6: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

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 mesh points $\mathbf{x}(i)$ are not ordered correctly,
or **npts** < 3,
or **npde** < 1,
or **nleft** is not in the range 0 to **npde**,
or **acc** ≤ 0.0,
or **ind** $\neq 0$ or 1,
or **lrsave** is too small,
or **lisave** is too small,
or d03pe called initially with **ind** $= 1$.

ifail $= 2$

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 $= 3$

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. Incorrect positioning of boundary conditions may also result in this error. 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 the user-supplied (sub)program **pdedef** or user-supplied (sub)program **bndary**, when the residual in the underlying ODE solver was being evaluated. Incorrect positioning of boundary conditions may also result in this error.

ifail $= 5$

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

ifail $= 6$

When evaluating the residual in solving the ODE system, **ires** was set to 2 in one of the user-supplied (sub)programs **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 the user-supplied (sub)programs , **pdedef** or **bndary**, **ires** was set to an invalid value.

ifail $= 9$ (d02nn)

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

ifail = 10

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 advisory message unit).

7 Accuracy

d03pe 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 parameter, **acc**.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first-order by the introduction of new variables (see the example problem in d03pk). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (d03pc or d03ph for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretization scheme with upwind weighting (d03pf for example), or the addition of a second-order artificial dissipation term.

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

9 Example

d03pe_bndary.m

```
function [res, ires] = bndary(npde, t, ibnd, nobc, u, ut, ires)
      if (ibnd == 0)
         if (ires == -1)
            res(1) = 0;
         else
             res(1) = u(1) - 0.5*(exp(t)+exp(-3*t)) - 0.25*(sin(-3*t)-
sin(t));
         end
      else
         if (ires == -1)
            res(1) = 0;
         else
                res(1) = u(2) - exp(1-3*t) + exp(1+t) - 0.5*(sin(1-
3*t)+sin(1+t));
         end
      end
```

d03pe_pdedef.m

```
function [res, ires] = pdedef(npde, t, x, u, ut, ux, ires)

      if (ires == -1)
```

```
         res(1) = ut(1);
         res(2) = ut(2);
      else
         res(1) = ut(1) + ux(1) + ux(2);
         res(2) = ut(2) + 4.0d0*ux(1) + ux(2);
      end
```

```
ts = 0;
tout = 0.2;
u = [1, 1.025315120524429, 1.051271096376024, 1.077884150884632, ...
    1.105170918075648, 1.133148453066826, 1.161834242728283, ...
    1.191246216612358, 1.22140275816017, 1.252322716191864, ...
    1.284025416687741, 1.316530674867622, 1.349858807576003, ...
    1.384030645980751, 1.419067548593257, 1.454991414618201, ...
    1.49182469764127, 1.529590419663379, 1.568312185490169, ...
    1.608014197485783, 1.648721270700128, 1.690458848379091, ...
    1.733253017867395, 1.777130526914038, 1.822118800390509, ...
    1.868245957432222, 1.915540829013896, 1.964032975969847, ...
    2.013752707470477, 2.064731099966486, 2.117000016612675, ...
    2.170592127183443, 2.225540928492468, 2.281880765329304, ...
    2.339646851925991, 2.398875293967098, 2.45960311115695, ...
    2.521868260358148, 2.585709659315846, 2.651167210982607, ...
    2.718281828459045;
       0, 0.02499739591471233, 0.04997916927067833, 0.07492970727274234,
...
    0.09983341664682815, 0.1246747333852277, 0.1494381324735992, ...
    0.1741081375935959, 0.1986693307950612, 0.2231063621317455, ...
    0.2474039592545229, 0.2715469369561129, 0.2955202066613395, ...
    0.319308785857001, 0.3428978074554513, 0.3662725290860476, ...
    0.3894183423086505, 0.4123207817434247, 0.4349655341112302, ...
    0.4573384471789554, 0.479425538604203, 0.5012130046737979, ...
    0.5226872289306592, 0.5438347906836426, 0.5646424733950354, ...
    0.5850972729404622, 0.6051864057360395, 0.6248973167276999, ...
    0.644217687237691, 0.6631354426633497, 0.6816387600233341, ...
    0.6997160753466035, 0.7173560908995228, 0.7345477822465786, ...
    0.7512804051402927, 0.7675435022360271, 0.7833269096274834, ...
    0.7986207631988143, 0.8134155047893737, 0.8277018881672576, ...
    0.8414709848078965];
x = [0;
     0.025;
     0.05;
     0.075;
     0.1;
     0.125;
     0.15;
     0.175;
     0.2;
     0.225;
     0.25;
     0.275;
     0.3;
     0.325;
     0.35;
     0.375;
     0.4;
     0.425;
     0.45;
     0.475;
     0.5;
     0.525;
     0.55;
     0.575;
     0.6;
     0.625;
     0.65;
     0.675;
     0.7;
     0.725;
```

```
     0.75;
     0.775;
     0.8;
     0.825;
     0.85;
     0.875;
     0.9;
     0.925;
     0.95;
     0.975;
     1];
nleft = int32(1);
acc = 1e-06;
rsave = zeros(2281, 1);
isave = zeros(106, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
[tsOut, uOut, rsaveOut, isaveOut, indOut, ifail] = ...
    d03pe(ts, tout, 'd03pe_pdedef', 'd03pe_bndary', u, x, nleft, acc, ...
    rsave, isave, itask, itrace, ind)
```

```
tsOut =
    0.2000
uOut =
  Columns 1 through 7
    0.6943    0.7158    0.7380    0.7609    0.7845    0.8088    0.8339
   -0.8555   -0.8499   -0.8447   -0.8398   -0.8352   -0.8311   -0.8274
  Columns 8 through 14
    0.8597    0.8864    0.9138    0.9420    0.9711    1.0010    1.0318
   -0.8242   -0.8215   -0.8192   -0.8176   -0.8164   -0.8159   -0.8160
  Columns 15 through 21
    1.0635    1.0961    1.1296    1.1640    1.1995    1.2359    1.2733
   -0.8168   -0.8182   -0.8204   -0.8233   -0.8269   -0.8314   -0.8367
  Columns 22 through 28
    1.3117    1.3512    1.3918    1.4334    1.4762    1.5201    1.5652
   -0.8428   -0.8499   -0.8579   -0.8668   -0.8767   -0.8877   -0.8997
  Columns 29 through 35
    1.6115    1.6590    1.7078    1.7578    1.8091    1.8618    1.9158
   -0.9128   -0.9270   -0.9424   -0.9590   -0.9768   -0.9958   -1.0162
  Columns 36 through 41
    1.9713    2.0281    2.0864    2.1462    2.2075    2.2703
   -1.0379   -1.0609   -1.0854   -1.1113   -1.1387   -1.1676
rsaveOut =
     array elided
isaveOut =
     array elided
indOut =
           1
ifail =
           0
```