# NAG Toolbox for MATLAB

# d03pc

## 1 Purpose

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, ifail] = d03pc(m, ts, tout, pdedef, bndary,
u, x, acc, rsave, isave, itask, itrace, ind, 'npde', npde, 'npts', npts,
'lrsave', lrsave, 'lisave', lisave)
```

## 3 Description

d03pc integrates the system of parabolic equations:

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

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 $b = x_{\mathbf{npts}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathbf{npts}}$. The co-ordinate system in space is defined by the value of $m$; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates. 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 a user-supplied (sub)program **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

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

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

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

The problem is subject to the following restrictions:

(i) $t_0 < t_{\mathrm{out}}$, 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 user-supplied (sub)program **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 8.

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 co-ordinates, 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* (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

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 – int32 scalar**

The co-ordinate system used:

**m** $= 0$

Indicates Cartesian co-ordinates.

**m** $= 1$

Indicates cylindrical polar co-ordinates.

**m** $= 2$

Indicates spherical polar co-ordinates.

*Constraint*: $0 \leq$ **m** $\leq 2$.

2: **ts – double scalar**

The initial value of the independent variable $t$.

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

3: **tout – double scalar**

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

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

**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 d03pc.

```
[p, q, r, ires] = pdedef(npde, t, x, u, 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: **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**.

6: **ires – int32 scalar**

Set to $-1$ or 1.

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 d03pc returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

**Output Parameters**

1: **p(npde,npde) – double array**

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

2: **q(npde) – double 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) – double 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 – int32 scalar**

Set to $-1$ or 1.

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 d03pc returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

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

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

```
[beta, gamma, ires] = bndary(npde, t, u, ux, ibnd, 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: **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**.

4: **ux(npde) – double 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 – int32 scalar**

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 – int32 scalar**

Set to $-1$ or 1.

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 d03pc returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

**Output Parameters**

1: **beta**(**npde**) – **double 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**) – **double 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** – **int32 scalar**

Set to $-1$ or 1.

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 d03pc returns to the calling (sub)program with the error indicator set to **ifail** $= 4$.

6: **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**.

7: **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**).

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 $j$th 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**) – **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** $= 2$

One step and return.

**itask** $= 3$

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

*Constraint*: $1 \leq$ **itask** $\leq 3$.

12: **itrace** – **int32 scalar**

The level of trace information required from 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 x04aa).

**itrace** $> 0$

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.

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 – 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 d03pc.

*Constraint*: $0 \leq$ **ind** $\leq 1$.

## 5.2 Optional Input Parameters

1: **npde – int32 scalar**

the number of PDEs in the system to be solved.

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

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** $\geq 3$.

3: **lrsave – int32 scalar**

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

*Constraint*: **lrsave** $\geq (6 \times$ **npde** $+ 10) \times$ **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 **m** $\neq 0$, 1 or 2,
or **m** $> 0$ and $\mathbf{x}(1) < 0.0$,
or the mesh points $\mathbf{x}(i)$ are not ordered,
or **npts** $< 3$,
or **npde** $< 1$,
or **acc** $\leq 0.0$,
or **ind** $\neq 0$ or 1,
or **lrsave** is too small,
or **lisave** is too small.

**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. 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 one of the user-supplied (sub)programs **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

When evaluating the residual in solving the ODE system, **ires** was set to 2 in at least 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 the 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 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.

## 7 Accuracy

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 parameter, **acc**.

## 8 Further Comments

d03pc is designed to solve parabolic systems (possibly including some elliptic equations) with second-order derivatives in space. The parameter 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 d03pe.

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

## 9 Example

d03pc_bndary.m

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

d03pc_pdedef.m

```
function [p, q, r, ires, user] = pdedef(npde, t, x, u, ux, ires, user)
      p = zeros(npde,npde);
      q = zeros(npde,1);
      r = zeros(npde,1);
      alpha = 1;
      q(1) = 4*alpha*(u(2)+x*ux(2));
      q(2) = 0;
      r(1) = x*ux(1);
      r(2) = ux(2) - u(1)*u(2);
      p(1,1) = 0;
      p(1,2) = 0;
      p(2,1) = 0;
      p(2,2) = 1 - x*x;
```

```
m = int32(1);
ts = 0;
tout = 0.0001;
u = [0, 0.1651586909446646, 0.3291891805614678, 0.4909709742815982, ...
    0.6493989384093669, 0.8033908493059388, 0.951894786074147, ...
    1.093896316244854, 1.228425425379336, 1.354563143251482, ...
    1.471447821346263, 1.578281018792787, 1.674332956525057, ...
    1.758947502412978, 1.831546653310115, 1.891634483401269, ...
    1.938800531878661, 1.972722606805445, 1.99316898601334, 2;
     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
x = [0;
     0.08257934547233232;
     0.1645945902807339;
     0.2454854871407991;
     0.3246994692046835;
     0.4016954246529694;
     0.4759473930370735;
     0.5469481581224268;
```

```
     0.6142127126896678;
     0.6772815716257411;
     0.7357239106731316;
     0.7891405093963936;
     0.8371664782625285;
     0.879473751206489;
     0.9157733266550574;
     0.9458172417006346;
     0.9694002659393304;
     0.9863613034027223;
     0.9965844930066698;
     1];
acc = 0.001;
rsave = zeros(1128, 1);
isave = zeros(64, 1, 'int32');
itask = int32(1);
itrace = int32(0);
ind = int32(0);
cwsav = {'', '', '', '', '', '', '', '', '', ''};
lwsav = false(100, 1);
iwsav = zeros(505, 1, 'int32');
rwsav = zeros(1100, 1);
[tsOut, uOut, rsaveOut, isaveOut, indOut, user, cwsavOut, ...
 lwsavOut, iwsavOut, rwsavOut, ifail] = ...
    d03pc(m, ts, tout, 'd03pc_pdedef', 'd03pc_bndary', u, x, acc, ...
    rsave, isave, itask, itrace, ind, cwsav, lwsav, iwsav, rwsav)
```

```
tsOut =
   1.0000e-04
uOut =
  Columns 1 through 7
         0    0.1760    0.3342    0.4938    0.6511    0.8045    0.9525
    0.9997    0.9996    0.9996    0.9996    0.9996    0.9995    0.9995
  Columns 8 through 14
    1.0942    1.2285    1.3544    1.4711    1.5778    1.6736    1.7576
    0.9994    0.9994    0.9993    0.9991    0.9989    0.9985    0.9936
  Columns 15 through 20
    1.8258    1.8678    1.8784    1.8684    1.8545    1.8485
    0.9449    0.7493    0.4582    0.2084    0.0525         0
rsaveOut =
     array elided
isaveOut =
     array elided
indOut =
           1
user =
     0
cwsavOut =
    [1x80 char]
    ''
    [1x80 char]
    [1x80 char]
    [1x80 char]
    ''
    ''
    ''
    ''
    ''
lwsavOut =
     array elided
iwsavOut =
     array elided
rwsavOut =
     array elided
ifail =
           0
```