NAG Toolbox for MATLAB

d03ra

1 Purpose

d03ra integrates a system of linear or nonlinear, time-dependent partial differential equations (PDEs) in two space dimensions on a rectangular domain. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs) which are solved using a backward differentiation formula (BDF) method. The resulting system of nonlinear equations is solved using a modified Newton method and a Bi-CGSTAB iterative linear solver with ILU preconditioning. Local uniform grid refinement is used to improve the accuracy of the solution. d03ra originates from the VLUGR2 package (see Blom and Verwer 1993 and Blom *et al.* 1996).

2 Syntax

```
[ts, dt, rwk, iwk, ind, ifail] = d03ra(ts, tout, dt, xmin, xmax, ymin,
ymax, nx, ny, tols, tolt, pdedef, bndary, pdeiv, monitr, opti, optr,
rwk, iwk, itrace, ind, 'npde', npde, 'lenrwk', lenrwk, 'leniwk', leniwk)
```

3 Description

d03ra integrates the system of PDEs:

$$F_j\left(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}\right)=0, \qquad j=1,2,\ldots,\mathbf{npde}, \tag{1}$$

for x and y in the rectangular domain $x_{\min}\le x\le x_{\max}$, $y_{\min}\le y\le y_{\max}$, and time interval $t_0\le t\le t_{\text{out}}$, where the vector u is the set of solution values

$$u(x,y,t)=\left[u_1(x,y,t),\ldots,u_{\mathbf{npde}}(x,y,t)\right]^{\mathrm{T}},$$

and u_t denotes partial differentiation with respect to t, and similarly for u_x etc.

The functions F_j must be supplied by you in a user-supplied (sub)program **pdedef**. Similarly the initial values of the functions $u(x,y,t)$ must be specified at $t=t_0$ in a user-supplied (sub)program **pdeiv**.

Note that whilst complete generality is offered by the master equations (1), d03ra is not appropriate for all PDEs. In particular, hyperbolic systems should not be solved using this function. Also, at least one component of u_t must appear in the system of PDEs.

The boundary conditions must be supplied by you in a user-supplied (sub)program **bndary** in the form

$$G_j\left(t,x,y,u,u_t,u_x,u_y\right)=0 \qquad \text{at } x=x_{\min},x_{\max},y=y_{\min},y_{\max}, \qquad j=1,2,\ldots,\mathbf{npde}. \tag{2}$$

The domain is covered by a uniform coarse base grid of size $n_x\times n_y$ specified by you, and nested finer uniform subgrids are subsequently created in regions with high spatial activity. The refinement is controlled using a space monitor which is computed from the current solution and a user-supplied space tolerance **tols**. A number of optional parameters, e.g., the maximum number of grid levels at any time, and some weighting factors, can be specified in the arrays **opti** and **optr**. Further details of the refinement strategy can be found in Section 8.

The system of PDEs and the boundary conditions are discretized in space on each grid using a standard second-order finite difference scheme (centred on the internal domain and one-sided at the boundaries), and the resulting system of ODEs is integrated in time using a second-order, two-step, implicit BDF method with variable step size. The time integration is controlled using a time monitor computed at each grid level from the current solution and a user-supplied time tolerance **tolt**, and some further optional user-specified weighting factors held in **optr** (see Section 8 for details). The time monitor is used to compute a new step size, subject to restrictions on the size of the change between steps, and (optional) user-specified maximum and minimum step sizes held in **dt**. The step size is adjusted so that the remaining integration interval is an integer number times Δt. In this way a solution is obtained at $t=t_{\text{out}}$.

A modified Newton method is used to solve the nonlinear equations arising from the time integration. You may specify (in **opti**) the maximum number of Newton iterations to be attempted. A Jacobian matrix is calculated at the beginning of each time step. If the Newton process diverges or the maximum number of iterations is exceeded, a new Jacobian is calculated using the most recent iterates and the Newton process is restarted. If convergence is not achieved after the (optional) user-specified maximum number of new Jacobian evaluations, the time step is retried with $\Delta t = \Delta t/4$. The linear systems arising from the Newton iteration are solved using a Bi-CGSTAB iterative method, in combination with ILU preconditioning. The maximum number of iterations can be specified by you in **opti**.

The solution at all grid levels is stored in the workspace arrays, along with other information needed for a restart (i.e., a continuation call). It is not intended that you extract the solution from these arrays, indeed the necessary information regarding these arrays is not included. The user-supplied monitor user-supplied (sub)program **monitr** should be used to obtain the solution at particular levels and times. **monitr** is called at the end of every time step, with the last step being identified via the input parameter **tlast**.

Within the user-supplied (sub)programs **pdeiv**, **pdedef**, **bndary** and **monitr** the data structure is as follows. Each point on a particular grid is given an index (ranging from 1 to the total number of points on the grid) and all co-ordinate or solution information is stored in arrays according to this index, e.g., $\mathbf{x}(i)$ and $\mathbf{y}(i)$ contain the x- and y co-ordinate of point i, and $\mathbf{u}(i,j)$ contains the jth solution component u_j at point i.

Further details of the underlying algorithm can be found in Section 8 and in Blom and Verwer 1993 and Blom *et al.* 1996 and the references therein.

4 References

Adjerid S and Flaherty J E 1988 A local refinement finite element method for two dimensional parabolic systems *SIAM J. Sci. Statist. Comput.* **9** 792–811

Blom J G, Trompert R A and Verwer J G 1996 Algorithm 758. VLUGR2: A vectorizable adaptive grid solver for PDEs in 2D *Trans. Math. Software* **22** 302–328

Blom J G and Verwer J G 1993 VLUGR2: A vectorized local uniform grid refinement code for PDEs in 2D *Report NM-R9306* CWI, Amsterdam

Brown P N, Hindmarsh A C and Petzold L R 1994 Using Krylov methods in the solution of large scale differential-algebraic systems *SIAM J. Sci. Statist. Comput.* **15** 1467–1488

Trompert R A 1993 Local uniform grid refinement and systems of coupled partial differential equations *Appl. Numer. Maths* **12** 331–355

Trompert R A and Verwer J G 1993 Analysis of the implicit Euler local uniform grid refinement method *SIAM J. Sci. Comput.* **14** 259–278

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: **dt**(**3**) **– double array**

The initial, minimum and maximum time step sizes respectively.

dt(1)

Specifies the initial time step size to be used on the first entry, i.e., when **ind** $= 0$. If **dt**$(1) = 0.0$ then the default value **dt**$(1) = 0.01 \times$ (**tout** $-$ **ts**) is used. On subsequent entries (**ind** $= 1$), the value of **dt**(1) is not referenced.

dt(2)

Specifies the minimum time step size to be attempted by the integrator. If **dt**$(2) = 0.0$ the default value **dt**$(2) = 10.0 \times$ ***machine precision*** is used.

dt(3)

Specifies the maximum time step size to be attempted by the integrator. If **dt**$(3) = 0.0$ the default value **dt**$(3) =$ **tout** $-$ **ts** is used.

Constraints:

if **ind** $= 0$, **dt**$(1) \geq 0$;
if **ind** $= 0$ and **dt**$(1) > 0$, $10.0 \times$ ***machine precision*** $\times \max(|\mathbf{ts}|, |\mathbf{tout}|) \leq$ **dt**$(1) \leq$ **tout** $-$ **ts** and **dt**$(2) \leq$ **dt**$(1) \leq$ **dt**(3), where the values of **dt**(2) and **dt**(3) will have been reset to their default values if zero on entry;
$0 \leq$ **dt**$(2) \leq$ **dt**(3).

4: **xmin – double scalar**

5: **xmax – double scalar**

The extents of the rectangular domain in the x-direction, i.e., the x co-ordinates of the left and right boundaries respectively.

Constraint: **xmin** $<$ **xmax** and **xmax** must be sufficiently distinguishable from **xmin** for the precision of the machine being used.

6: **ymin – double scalar**

7: **ymax – double scalar**

The extents of the rectangular domain in the y-direction, i.e., the y co-ordinates of the lower and upper boundaries respectively.

Constraint: **ymin** $<$ **ymax** and **ymax** must be sufficiently distinguishable from **ymin** for the precision of the machine being used.

8: **nx – int32 scalar**

The number of grid points in the x-direction (including the boundary points).

Constraint: **nx** ≥ 4.

9: **ny – int32 scalar**

The number of grid points in the y-direction (including the boundary points).

Constraint: **ny** ≥ 4.

10: **tols – double scalar**

The space tolerance used in the grid refinement strategy (σ in equation (4)). See Section 8.2.

Constraint: **tols** > 0.0.

11: **tolt – double scalar**

The time tolerance used to determine the time step size (τ in equation (7)). See Section 8.3.

Constraint: **tolt** > 0.0.

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

pdedef must evaluate the functions F_j, for $j = 1, 2, \ldots,$ **npde**, in equation (1) which define the system of PDEs (i.e., the residuals of the resulting ODE system) at all interior points of the domain. Values at points on the boundaries of the domain are ignored and will be overwritten by the user-supplied (sub)program **bndary**. **pdedef** is called for each subgrid in turn.

Its specification is:

```
[res] = pdedef(npts, npde, t, x, y, u, ut, ux, uy, uxx, uxy, uyy)
```

Input Parameters

1: **npts – int32 scalar**

The number of grid points in the current grid.

2: **npde – int32 scalar**

The number of PDEs in the system.

3: **t – double scalar**

The current value of the independent variable t.

4: **x**(**npts**) **– double array**

x(i) contains the x co-ordinate of the ith grid point, for $i = 1, 2, \ldots,$ **npts**.

5: **y**(**npts**) **– double array**

y(i) contains the y co-ordinate of the ith grid point, for $i = 1, 2, \ldots,$ **npts**.

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

u(i,j) contains the value of the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

7: **ut**(**npts,npde**) **– double array**

ut(i,j) contains the value of $\dfrac{\partial u}{\partial t}$ for the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

8: **ux**(**npts,npde**) **– double array**

ux(i,j) contains the value of $\dfrac{\partial u}{\partial x}$ for the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

9: **uy**(**npts,npde**) **– double array**

uy(i,j) contains the value of $\dfrac{\partial u}{\partial y}$ for the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

10: **uxx**(**npts,npde**) **– double array**

uxx(i,j) contains the value of $\dfrac{\partial^2 u}{\partial x^2}$ for the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

11: **uxy**(**npts**,**npde**) – **double array**

uxy(i,j) contains the value of $\dfrac{\partial^2 u}{\partial x \partial y}$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

12: **uyy**(**npts**,**npde**) – **double array**

uyy(i,j) contains the value of $\dfrac{\partial^2 u}{\partial y^2}$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

Output Parameters

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

res(i,j) must contain the value of F_j, for $j = 1,2,\ldots,$ **npde**, at the ith grid point, for $i = 1,2,\ldots,$ **npts**, although the residuals at boundary points will be ignored (and overwritten later on) and so they need not be specified here.

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

bndary must evaluate the functions G_j, for $j = 1,2,\ldots,$ **npde**, in equation (2) which define the boundary conditions at all boundary points of the domain. Residuals at interior points must **not** be altered by this (sub)program.

Its specification is:

```
[res] = bndary(npts, npde, t, x, y, u, ut, ux, uy, nbpts, lbnd, res)
```

Input Parameters

1: **npts – int32 scalar**

The number of grid points in the current grid.

2: **npde – int32 scalar**

The number of PDEs in the system.

3: **t – double scalar**

The current value of the independent variable t.

4: **x**(**npts**) – **double array**

x(i) contains the x co-ordinate of the ith grid point, for $i = 1,2,\ldots,$ **npts**.

5: **y**(**npts**) – **double array**

y(i) contains the y co-ordinate of the ith grid point, for $i = 1,2,\ldots,$ **npts**.

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

u(i,j) contains the value of the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

7: **ut**(**npts**,**npde**) – **double array**

ut(i,j) contains the value of $\dfrac{\partial u}{\partial t}$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

8: **ux**(**npts**,**npde**) – **double array**

ux(i,j) contains the value of $\dfrac{\partial u}{\partial x}$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

9: **uy**(**npts**,**npde**) – **double array**

uy(i,j) contains the value of $\dfrac{\partial u}{\partial y}$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$ **npts** and $j = 1,2,\ldots,$ **npde**.

10: **nbpts** – **int32 scalar**

The number of boundary points in the grid.

11: **lbnd**(**nbpts**) – **int32 array**

lbnd(i) contains the grid index for the ith boundary point for $i = 1,2,\ldots,$ **nbpts**. Hence the ith boundary point has co-ordinates **x**(**lbnd**(i)) and **y**(**lbnd**(i)), and the corresponding solution values are **u**(**lbnd**(i)**npde**), etc.

12: **res**(**npts**,**npde**) – **double array**

res(i,j) contains the value of F_j, for $i = 1,2,\ldots,$ **npde**, at the ith grid point for $i = 1,2,\ldots,$ **npts**, as returned by user-supplied (sub)program **pdedef**. The residuals at the boundary points will be overwritten and so need not have been set by **pdedef**.

res(**lbnd**$(i)j$) must contain the value of G_j, for $j = 1,2,\ldots,$ **npde**, at the ith boundary point for $i = 1,2,\ldots,$ **nbpts**.

Note: elements of **res** corresponding to interior points must **not** be altered.

Output Parameters

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

res(i,j) contains the value of F_j, for $i = 1,2,\ldots,$ **npde**, at the ith grid point for $i = 1,2,\ldots,$ **npts**, as returned by user-supplied (sub)program **pdedef**. The residuals at the boundary points will be overwritten and so need not have been set by **pdedef**.

res(**lbnd**$(i)j$) must contain the value of G_j, for $j = 1,2,\ldots,$ **npde**, at the ith boundary point for $i = 1,2,\ldots,$ **nbpts**.

Note: elements of **res** corresponding to interior points must **not** be altered.

14: **pdeiv** – **string containing name of m-file**

pdeiv must specify the initial values of the PDE components u at all points in the grid. **pdeiv** is not referenced if, on entry, **ind** $= 1$.

Its specification is:

```
[u] = pdeiv(npts, npde, t, x, y)
```

Input Parameters

1: **npts** – **int32 scalar**

The number of grid points in the grid.

2: **npde** – **int32 scalar**

The number of PDEs in the system.

3: **t – double scalar**

The (initial) value of the independent variable t.

4: **x(npts) – double array**

x(i) contains the x co-ordinate of the ith grid point, for $i = 1, 2, \ldots,$ **npts**.

5: **y(npts) – double array**

y(i) contains the y co-ordinate of the ith grid point, for $i = 1, 2, \ldots,$ **npts**.

Output Parameters

1: **u(npts,npde) – double array**

u(i,j) must contain the value of the jth PDE component at the ith grid point, for $i = 1, 2, \ldots,$ **npts** and $j = 1, 2, \ldots,$ **npde**.

15: **monitr – string containing name of m-file**

monitr is called by d03ra at the end of every successful time step, and may be used to examine or print the solution or perform other tasks such as error calculations, particularly at the final time step, indicated by the parameter **tlast**. The input arguments contain information about the grid and solution at all grid levels used.

monitr can also be used to force an immediate tidy termination of the solution process and return to the calling program.

Its specification is:

```
[ierr] = monitr(npde, t, dt, dtnew, tlast, nlev, ngpts, xpts, ypts,
lsol, sol, ierr)
```

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, i.e., the time at the end of the integration step just completed.

3: **dt – double scalar**

The current time step size Δt, i.e., the time step size used for the integration step just completed.

4: **dtnew – double scalar**

The step size that will be used for the next time step.

5: **tlast – logical scalar**

Indicates if intermediate or final time step. **tlast** = **false** for an intermediate step, **tlast** = **true** for the last call to **monitr** before returning to your program.

6: **nlev – int32 scalar**

The number of grid levels used at time **t**.

7: **ngpts**(**nlev**) – **int32 array**

ngpts(l) contains the number of grid points at level l, for $l = 1, 2, \ldots,$ **nlev**.

8: **xpts**($lpts$) – **double array**

Contains the x co-ordinates of the grid points in each level in turn, i.e., **x**(i), for $i = 1, 2, \ldots,$ **ngpts**(l), for $l = 1, 2, \ldots,$ **nlev**.

So for level l, **x**(i) = **xpts**($k + i$), where $k =$ **ngpts**(1) + **ngpts**(2) + $\cdots$ + **ngpts**($l - 1$), for $i = 1, 2, \ldots,$ **ngpts**(l) and $l = 1, 2, \ldots,$ **nlev**.

9: **ypts**($lpts$) – **double array**

Contains the y co-ordinates of the grid points in each level in turn, i.e., **y**(i), for $i = 1, 2, \ldots,$ **ngpts**(l) and $l = 1, 2, \ldots,$ **nlev**.

So for level l, **y**(i) = **ypts**($k + i$), where $k =$ **ngpts**(1) + **ngpts**(2) + $\cdots$ + **ngpts**($l - 1$), for $i = 1, 2, \ldots,$ **ngpts**(l) and $l = 1, 2, \ldots,$ **nlev**.

10: **lsol**(**nlev**) – **int32 array**

lsol(l) contains the pointer to the solution in **sol** at grid level l and time **t**. (**lsol**(l) actually contains the array index immediately preceding the start of the solution in **sol**.)

11: **sol**($lpts \times$ **npde**) – **double array**

Contains the solution **u**(**ngpts**(l)**npde**) at time **t** for each grid level l in turn, positioned according to **lsol**, i.e., for level l,

$$\mathbf{u}(i,j) = \mathbf{sol}(\mathbf{lsol}(l) + (j-1) \times \mathbf{ngpts}(l) + i),$$

for $i = 1, \ldots,$ **ngpts**(l), $j = 1, \ldots,$ **npde** and $l = 1, \ldots,$ **nlev**.

12: **ierr** – **int32 scalar**

May contain data passed by the function.

Should be set to 1 to force a tidy termination and an immediate return to the calling program with **ifail** = 4. **ierr** should remain unchanged otherwise.

Output Parameters

1: **ierr** – **int32 scalar**

May contain data passed by the function.

Should be set to 1 to force a tidy termination and an immediate return to the calling program with **ifail** = 4. **ierr** should remain unchanged otherwise.

16: **opti**(**4**) – **int32 array**

May be set to control various options available in the integrator.

opti$(1) = 0$

All the default options are employed.

opti$(1) > 0$

The default value of **opti**(i), for $i = 2, 3$ or 4, can be obtained by setting **opti**$(i) = 0$.

opti(1)

Specifies the maximum number of grid levels allowed (including the base grid). **opti**$(1) \geq 0$. The default value is **opti**$(1) = 3$.

opti(2)

Specifies the maximum number of Jacobian evaluations allowed during each nonlinear equations solution. **opti**$(2) \geq 0$. The default value is **opti**$(2) = 2$.

opti(3)

Specifies the maximum number of Newton iterations in each nonlinear equations solution. **opti**$(3) \geq 0$. The default value is **opti**$(3) = 10$.

opti(4)

Specifies the maximum number of iterations in each linear equations solution. **opti**$(4) \geq 0$. The default value is **opti**$(4) = 100$.

Constraint: **opti**$(1) \geq 0$ and if **opti**$(1) > 0$, **opti**$(i) \geq 0$, for $i = 2, 3$ or 4.

17: **optr**(**3,npde**) – **double array**

May be used to specify the optional vectors $u^{\max}$, w^{s} and w^{t} in the space and time monitors (see Section 8).

If an optional vector is not required then all its components should be set to 1.0.

optr$(1,j)$, for $j = 1, 2, \ldots,$ **npde**, specifies $u_j^{\max}$, the approximate maximum absolute value of the jth component of u, as used in (4) and (7). **optr**$(1,j) > 0.0$, for $j = 1, 2, \ldots,$ **npde**.

optr$(2,j)$, for $j = 1, 2, \ldots,$ **npde**, specifies w_j^{s}, the weighting factors used in the space monitor (see (4)) to indicate the relative importance of the jth component of u on the space monitor. **optr**$(2,j) \geq 0.0$, for $j = 1, 2, \ldots,$ **npde**.

optr$(3,j)$, for $j = 1, 2, \ldots,$ **npde**, specifies w_j^{t}, the weighting factors used in the time monitor (see (6)) to indicate the relative importance of the jth component of u on the time monitor. **optr**$(3,j) \geq 0.0$, for $j = 1, 2, \ldots,$ **npde**.

Constraints:

optr$(1, j) > 0.0$, for $j = 1, 2, \ldots,$ **npde**;
optr$(i, j) \geq 0.0$, for $i = 2, 3$ and $j = 1, 2, \ldots,$ **npde**.

18: **rwk**(**lenrwk**) – **double array**

The required value of **lenrwk** cannot be determined exactly in advance, but a suggested value is

$$\textbf{lenrwk} = maxpts \times \textbf{npde} \times (5 \times l + 18 \times \textbf{npde} + 9) + 2 \times maxpts,$$

where $l =$ **opti**(1) if **opti**$(1) \neq 0$ and $l = 3$ otherwise, and $maxpts$ is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the function returns with **ifail** $= 3$ and an estimated required size is printed on the current error message unit (see x04aa).

Constraint: **lenrwk** $\geq$ **nx** $\times$ **ny** $\times$ **npde** $\times (14 + 18 \times$ **npde**$) + 2 \times$ **nx** $\times$ **ny** (the required size for the initial grid).

19: **iwk**(**leniwk**) – **int32 array**

If **ind** $= 0$, **iwk** need not be set. Otherwise **iwk** must remain unchanged from a previous call to d03ra.

20: **itrace** – **int32 scalar**

The level of trace information required from d03ra. **itrace** may take the value -1, 0, 1, 2 or 3.

itrace $= -1$

No output is generated.

itrace $= 0$

Only warning messages are printed.

itrace > 0

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

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. Setting **itrace** $= 1$ allows you to monitor the progress of the integration without possibly excessive information.

21: **ind** – **int32 scalar**

Must be set to 0 or 1.

ind $= 0$

Starts the integration in time.

ind $= 1$

Continues the integration after an earlier exit from the function. In this case, only the following parameters may be reset between calls to d03ra: **tout**, **dt**(2), **dt**(3), **tols**, **tolt**, **opti**, **optr**, **itrace** and **ifail**.

Constraint: $0 \leq$ **ind** ≤ 1.

5.2 Optional Input Parameters

1: **npde** – **int32 scalar**

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

the number of PDEs in the system.

Constraint: **npde** ≥ 1.

2: **lenrwk** – **int32 scalar**

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

The required value of **lenrwk** cannot be determined exactly in advance, but a suggested value is

$$\textbf{lenrwk} = maxpts \times \textbf{npde} \times (5 \times l + 18 \times \textbf{npde} + 9) + 2 \times maxpts,$$

where $l =$ **opti**(1) if **opti**$(1) \neq 0$ and $l = 3$ otherwise, and $maxpts$ is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the function returns with **ifail** $= 3$ and an estimated required size is printed on the current error message unit (see x04aa).

Constraint: **lenrwk** $\geq$ **nx** $\times$ **ny** $\times$ **npde** $\times (14 + 18 \times$ **npde**$) + 2 \times$ **nx** $\times$ **ny** (the required size for the initial grid).

3: **leniwk – int32 scalar**

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

The required value of **leniwk** cannot be determined exactly in advance, but a suggested value is

$$\mathbf{leniwk} = maxpts \times (14 + 5 \times m) + 7 \times m + 2,$$

where $maxpts$ is the expected maximum number of grid points at any one level and $m = \mathbf{opti}(1)$ if $\mathbf{opti}(1) > 0$ and $m = 3$ otherwise. If during the execution the supplied value is found to be too small then the function returns with $\mathbf{ifail} = 3$ and an estimated required size is printed on the current error message unit (see x04aa).

Constraint: $\mathbf{leniwk} \geq 19 \times \mathbf{nx} \times \mathbf{ny} + 9$ (the required size for the initial grid).

5.3 Input Parameters Omitted from the MATLAB Interface

lwk, lenlwk

5.4 Output Parameters

1: **ts – double scalar**

The value of t which has been reached. Normally $\mathbf{ts} = \mathbf{tout}$.

2: **dt(3) – double array**

$\mathbf{dt}(1)$ contains the time step size for the next time step. $\mathbf{dt}(2)$ and $\mathbf{dt}(3)$ are unchanged or set to their default values if zero on entry.

3: **rwk(lenrwk) – double array**

4: **iwk(leniwk) – int32 array**

The following components of the array **iwk** concern the efficiency of the integration. Here, m is the maximum number of grid levels allowed ($m = \mathbf{opti}(1)$ if $\mathbf{opti}(1) > 1$ and $m = 3$ otherwise), and l is a grid level taking the values $l = 1, 2, \ldots, nl$, where nl is the number of levels used.

iwk(1)

Contains the number of steps taken in time.

iwk(2)

Contains the number of rejected time steps.

iwk$(2+l)$

Contains the total number of residual evaluations performed (i.e., the number of times user-supplied (sub)program **pdedef** was called) at grid level l.

iwk$(2+m+l)$

Contains the total number of Jacobian evaluations performed at grid level l.

iwk$(2+2\times m+l)$

Contains the total number of Newton iterations performed at grid level l.

iwk$(2+3\times m+l)$

Contains the total number of linear solver iterations performed at grid level l.

iwk$(2+4\times m+l)$

Contains the maximum number of Newton iterations performed at any one time step at grid level l.

iwk$(2+5\times m+l)$

Contains the maximum number of linear solver iterations performed at any one time step at grid level l.

Note: the total and maximum numbers are cumulative over all calls to d03ra. If the specified maximum number of Newton or linear solver iterations is exceeded at any stage, then the maximums above are set to the specified maximum plus one.

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, **npde** < 1,
or **tout** $\leq$ **ts**,
or **tout** is too close to **ts**,
or **ind** $= 0$ and **dt**$(1) < 0.0$,
or **dt**$(i) < 0.0$, for $i = 2$ or 3,
or **dt**$(2) >$ **dt**(3),
or **ind** $= 0.0$ and $0.0 <$ **dt**$(1) < 10 \times$ ***machine precision*** $\times \max(|\mathbf{ts}|, |\mathbf{tout}|)$,
or **ind** $= 0.0$ and **dt**$(1) >$ **tout** $-$ **ts**,
or **ind** $= 0.0$ and **dt**$(1) <$ **dt**(2) or **dt**$(1) >$ **dt**(3),
or **xmin** $\geq$ **xmax**,
or **xmax** too close to **xmin**,
or **ymin** $\geq$ **ymax**,
or **ymax** too close to **ymin**,
or **nx** or **ny** < 4,
or **tols** or **tolt** ≤ 0.0,

or **opti**$(1) < 0$,
or **opti**$(1) > 0$ and **opti**$(j) < 0$, for $j = 2, 3$ or 4,
or **optr**$(1,j) \leq 0.0$, for some $j = 1, 2, \ldots,$ **npde**,
or **optr**$(2,j) < 0.0$, for some $j = 1, 2, \ldots,$ **npde**,
or **optr**$(3,j) < 0.0$, for some $j = 1, 2, \ldots,$ **npde**,
or **lenrwk**, **leniwk** or **lenlwk** too small for initial grid level,
or **ind** $\neq 0$ or 1,
or **ind** $= 1$ on initial entry to d03ra.

ifail $= 2$

The time step size to be attempted is less than the specified minimum size. This may occur following time step failures and subsequent step size reductions caused by one or more of the following:

the requested accuracy could not be achieved, i.e., **tolt** is too small,

the maximum number of linear solver iterations, Newton iterations or Jacobian evaluations is too small,

ILU decomposition of the Jacobian matrix could not be performed, possibly due to singularity of the Jacobian.

Setting **itrace** to a higher value may provide further information.

In the latter two cases you are advised to check their problem formulation in user-supplied (sub)program **pdedef** and/or user-supplied (sub)program **bndary**, and the initial values in user-supplied (sub)program **pdeiv** if appropriate.

ifail $= 3$

One or more of the workspace arrays is too small for the required number of grid points. An estimate of the required sizes for the current stage is output, but more space may be required at a later stage.

ifail $= 4$

ierr was set to 1 in the user-supplied (sub)program **monitr**, forcing control to be passed back to calling program. Integration was successful as far as **t** $=$ **ts**.

ifail $= 5$

The integration has been completed but the maximum number of levels specified in **opti**(1) was insufficient at one or more time steps, meaning that the requested space accuracy could not be achieved. To avoid this warning either increase the value of **opti**(1) or decrease the value of **tols**.

7 Accuracy

There are three sources of error in the algorithm: space and time discretization, and interpolation (linear) between grid levels. The space and time discretization errors are controlled separately using the parameters **tols** and **tolt** described in the following section, and you should test the effects of varying these parameters. Interpolation errors are generally implicitly controlled by the refinement criterion since in areas where interpolation errors are potentially large, the space monitor will also be large. It can be shown that the global spatial accuracy is comparable to that which would be obtained on a uniform grid of the finest grid size. A full error analysis can be found in Trompert and Verwer 1993.

8 Further Comments

8.1 Algorithm Outline

The local uniform grid refinement method is summarized as follows:

1. Initialize the course base grid, an initial solution and an initial time step.

2. Solve the system of PDEs on the current grid with the current time step.
3. If the required accuracy in space and the maximum number of grid levels have not yet been reached:
   (a) Determine new finer grid at forward time level.
   (b) Get solution values at previous time level(s) on new grid.
   (c) Interpolate internal boundary-values from old grid at forward time.
   (d) Get initial values for the Newton process at forward time.
   (e) Go to 2.
4. Update the coarser grid solution using the finer grid values.
5. Estimate error in time integration. If time error is acceptable advance time level.
6. Determine new step size then go to 2 with coarse base as current grid.

8.2 Refinement Strategy

For each grid point i a space monitor μ_i^{s} is determined by

$$\mu_i^{\mathrm{s}} = \max_{j=1,\mathbf{npde}} \left\{ \gamma_j \left(\left| \Delta x^2 \frac{\partial^2}{\partial x^2} u_j(x_i, y_i, t) \right| + \left| \Delta y^2 \frac{\partial^2}{\partial y^2} u_j(x_i, y_i, t) \right| \right) \right\}, \tag{3}$$

where Δx and Δy are the grid widths in the x and y directions; and x_i, y_i are the x and y co-ordinates at grid point i. The parameter γ_j is obtained from

$$\gamma_j = \frac{w_j^{\mathrm{s}}}{u_j^{\max} \sigma}, \tag{4}$$

where σ is the user-supplied space tolerance; w_j^s is a weighting factor for the relative importance of the jth PDE component on the space monitor; and $u_j^{\max}$ is the approximate maximum absolute value of the jth component. A value for σ must be supplied by you. Values for w_j^{s} and $u_j^{\max}$ must also be supplied but may be set to the value 1.0 if little information about the solution is known.

A new level of refinement is created if

$$\max_i \{\mu_i^{\mathrm{s}}\} > 0.9 \qquad \text{or} \qquad 1.0, \tag{5}$$

depending on the grid level at the previous step in order to avoid fluctuations in the number of grid levels between time steps. If (5) is satisfied then all grid points for which $\mu_i^s > 0.25$ are flagged and surrounding cells are quartered in size.

No derefinement takes place as such, since at each time step the solution on the base grid is computed first and new finer grids are then created based on the new solution. Hence derefinement occurs implicitly. See Section 8.1.

8.3 Time Integration

The time integration is controlled using a time monitor calculated at each level l up to the maximum level used, given by

$$\mu_l^t = \sqrt{\frac{1}{N} \sum_{j=1}^{\mathbf{npde}} w_j^t \sum_{i=1}^{\mathbf{ngpts}(l)} \left(\frac{\Delta t}{\alpha_{ij}} u_t(x_i, y_i, t) \right)^2} \tag{6}$$

where $\mathbf{ngpts}(l)$ is the total number of points on grid level l; $N = \mathbf{ngpts}(l) \times \mathbf{npde}$; Δt is the current time step; u_t is the time derivative of u which is approximated by first-order finite differences; w_j^{t} is the time equivalent of the space weighting factor w_j^{s}; and α_{ij} is given by

$$\alpha_{ij} = \tau \left(\frac{u_j^{\max}}{100} + |u(x_i, y_i, t)| \right) \tag{7}$$

where $u_j^{\max}$ is as before, and τ is the user-specified time tolerance.

An integration step is rejected and retried at all levels if

$$\max_l \{\mu_l^t\} > 1.0. \tag{8}$$

9 Example

d03ra_bndary.m

```
function [res] = bndary(npts, npde, t, x, y, u, ut, ux, uy, nbpts,
lbnd, res)

  tol = x02aj();
  tol = 10*tol;
  for i = 1:nbpts
    j = lbnd(i);
    if (abs(x(j)) <= tol)
      res(j,1) = ux(j,1);
    elseif (abs(x(j)-1) <= tol)
      res(j,1) = u(j,1) - 1;
    elseif (abs(y(j)) <= tol)
      res(j,1) = uy(j,1);
    elseif (abs(y(j)-1) <= tol)
      res(j,1) = u(j,1) - 1;
    end
  end
```

d03ra_monitr.m

```
function [ierr] = ...
     monitr(npde, t, dt, dtnew, tlast, nlev, ngpts, xpts, ypts, lsol,
sol, ierr)
```

d03ra_pdedef.m

```
function [res] = pdedef(npts, npde, t, x, y, u, ut, ux, uy, uxx, uxy,
uyy)
  res = zeros(npts, npde);

  alpha = 1;
  delta = 20;
  reac = 5;
  diff = 0.1;
  d = reac*exp(delta)/(alpha*delta);

  for i = 1:npts
    res(i,1) = ut(i,1) - ...
                   diff*(uxx(i,1)+uyy(i,1)) - d*(1+alpha-u(i,1))*exp(-
delta/u(i,1)) ;
  end
```

d03ra_pdeiv.m

```
function [u] = pdeiv(npts, npde, t, x, y)
  u = ones(npts, npde);
```

```
ts = 0;
tout = 0.24;
dt = [0.001;
     0;
     0];
```

```
xmin = 0;
xmax = 1;
ymin = 0;
ymax = 1;
nx = int32(21);
ny = int32(21);
tols = 0.5;
tolt = 0.01;
opti = zeros(4, 1, 'int32');
optr = [1; 1; 1];
rwk = zeros(88000, 1);
iwk = zeros(58023, 1, 'int32');
itrace = int32(0);
ind = int32(0);
[tsOut, dtOut, rwkOut, iwkOut, indOut, ifail] = ...
    d03ra(ts, tout, dt, xmin, xmax, ymin, ymax, nx, ny, tols, tolt, ...
    'd03ra_pdedef', 'd03ra_bndary', 'd03ra_pdeiv', 'd03ra_monitr', ...
    opti, optr, rwk, iwk, itrace, ind)
```

```
tsOut =
    0.2400
dtOut =
    0.0001
    0.0000
    0.2400
rwkOut =
     array elided
iwkOut =
     array elided
indOut =
           1
ifail =
           0
```