# NAG Library Routine Document

# D03RBF

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

## 1 Purpose

D03RBF integrates a system of linear or nonlinear, time-dependent partial differential equations (PDEs) in two space dimensions on a rectilinear domain. The method of lines is employed to reduce the NPDEs 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. D03RBF originates from the VLUGR2 package (see Blom and Verwer (1993) and Blom *et al.* (1996)).

## 2 Specification

```
SUBROUTINE D03RBF (NPDE, TS, TOUT, DT, TOLS, TOLT, INIDOM, PDEDEF,          &
                   BNDARY, PDEIV, MONITR, OPTI, OPTR, RWK, LENRWK, IWK,     &
                   LENIWK, LWK, LENLWK, ITRACE, IND, IFAIL)

INTEGER            NPDE, OPTI(4), LENRWK, IWK(LENIWK), LENIWK, LENLWK,      &
                   ITRACE, IND, IFAIL
REAL (KIND=nag_wp) TS, TOUT, DT(3), TOLS, TOLT, OPTR(3,NPDE),               &
                   RWK(LENRWK)
LOGICAL            LWK(LENLWK)
EXTERNAL           INIDOM, PDEDEF, BNDARY, PDEIV, MONITR
```

## 3 Description

D03RBF integrates the system of PDEs:

$$F_j\big(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}\big)=0,\quad j=1,2,\ldots,\text{NPDE},\quad (x,y)\in\Omega,\quad t_0\le t\le t_{\text{out}},\qquad (1)$$

where $\Omega$ is an arbitrary rectilinear domain, i.e., a domain bounded by perpendicular straight lines. If the domain is rectangular then it is recommended that D03RAF is used.

The vector $u$ is the set of solution values

$$u(x,y,t)=[u_1(x,y,t),\ldots,u_{\text{NPDE}}(x,y,t)]^{\text{T}},$$

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 PDEDEF. Similarly the initial values of the functions $u(x,y,t)$ for $(x,y)\in\Omega$ must be specified at $t=t_0$ in PDEIV.

Note that whilst complete generality is offered by the master equations (1), D03RBF is not appropriate for all PDEs. In particular, hyperbolic systems should not be solved using this routine. Also, at least one component of $u_t$ must appear in the system of PDEs.

The boundary conditions must be supplied by you in BNDARY in the form

$$G_j\big(t,x,y,u,u_t,u_x,u_y\big)=0,\quad j=1,2,\ldots,\text{NPDE},\quad (x,y)\in\partial\Omega,\quad t_0\le t\le t_{\text{out}}.\qquad (2)$$

The domain is covered by a uniform coarse base grid 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 9.

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 9 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 $\Delta t$. 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.

In order to define the base grid you must first specify a virtual uniform rectangular grid which contains the entire base grid. The position of the virtual grid in physical $(x, y)$ space is given by the $(x, y)$ coordinates of its boundaries. The number of points $n_x$ and $n_y$ in the $x$ and $y$ directions must also be given, corresponding to the number of columns and rows respectively. This is sufficient to determine precisely the $(x, y)$ coordinates of all virtual grid points. Each virtual grid point is then referred to by integer coordinates $(v_x, v_y)$, where $(0, 0)$ corresponds to the lower-left corner and $(n_x - 1, n_y - 1)$ corresponds to the upper-right corner. $v_x$ and $v_y$ are also referred to as the virtual column and row indices respectively.

The base grid is then specified with respect to the virtual grid, with each base grid point coinciding with a virtual grid point. Each base grid point must be given an index, starting from 1, and incrementing row-wise from the leftmost point of the lowest row. Also, each base grid row must be numbered consecutively from the lowest row in the grid, so that row 1 contains grid point 1.

As an example, consider the domain consisting of the two separate squares shown in Figure 1. The left-hand diagram shows the virtual grid and its integer coordinates (i.e., its column and row indices), and the right-hand diagram shows the base grid point indices and the base row indices (in brackets).

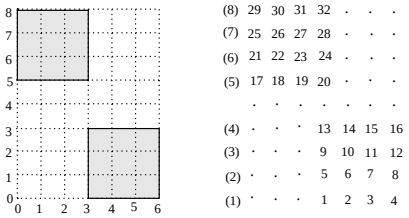


**Figure 1**

Hence the base grid point with index 6 say is in base row 2, virtual column 4, and virtual row 1, i.e., virtual grid integer coordinates $(4, 1)$; and the base grid point with index 19 say is in base row 5, virtual column 2, and virtual row 5, i.e., virtual grid integer coordinates $(2, 5)$.

The base grid must then be defined in INIDOM by specifying the number of base grid rows, the number of base grid points, the number of boundaries, the number of boundary points, and the following integer arrays:

LROW contains the base grid indices of the starting points of the base grid rows;

IROW contains the virtual row numbers $v_y$ of the base grid rows;

ICOL contains the virtual column numbers $v_x$ of the base grid points;

LBND contains the grid indices of the boundary edges (without corners) and corner points;

LLBND contains the starting elements of the boundaries and corners in LBND.

Finally, ILBND contains the types of the boundaries and corners, as follows:

Boundaries:

1 – lower boundary

2 – left boundary

3 – upper boundary

4 – right boundary

External corners (90°):

12 – lower-left corner

23 – upper-left corner

34 – upper-right corner

41 – lower-right corner

Internal corners (270°):

21 – lower-left corner

32 – upper-left corner

43 – upper-right corner

14 – lower-right corner

Figure 2 shows the boundary types of a domain with a hole. Notice the logic behind the labelling of the corners: each one includes the types of the two adjacent boundary edges, in a clockwise fashion (outside the domain).

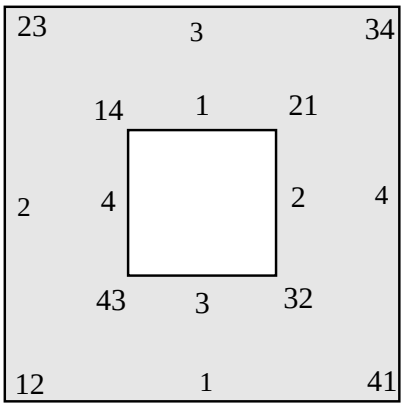


**Figure 2**

As an example, consider the domain shown in Figure 3. The left-hand diagram shows the physical domain and the right-hand diagram shows the base and virtual grids. The numbers outside the base grid are the indices of the left and rightmost base grid points, and the numbers inside the base grid are the boundary or corner numbers, indicating the order in which the boundaries are stored in LBND.

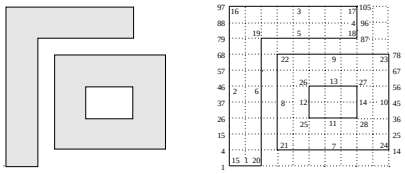

**Figure 3**

For this example we have

```
NROWS = 11
 NPTS = 105
```

```
      NBNDS = 28
      NBPTS = 72

      LROW = (1,4,15,26,37,46,57,68,79,88,97)

      IROW = (0,1,2,3,4,5,6,7,8,9,10)

      ICOL = (0,1,2,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,8,9,10,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,6,7,8,9,10,
              0,1,2,3,4,5,6,7,8,
              0,1,2,3,4,5,6,7,8,
              0,1,2,3,4,5,6,7,8)

      LBND = (2,
              4,15,26,37,46,57,68,79,88,
              98,99,100,101,102,103,104,
              96,
              86,85,84,83,82,
              70,59,48,39,28,17,6,
              8,9,10,11,12,13,
              18,29,40,49,60,
              72,73,74,75,76,77,
              67,56,45,36,25,
              33,32,
              42,
              52,53,
              43,
              1,97,105,87,81,3,7,71,78,14,31,51,54,34)

      LLBND = (1,2,11,18,19,24,31,37,42,48,53,55,56,58,59,60,
               61,62,63,64,65,66,67,68,69,70,71,72)

      ILBND = (1,2,3,4,1,4,1,2,3,4,3,4,1,2,12,23,34,41,14,41,
               12,23,34,41,43,14,21,32)
```

This particular domain is used in the example in Section 10, and data statements are used to define the above arrays in that example program. For less complicated domains it is simpler to assign the values of the arrays in do-loops. This also allows flexibility in the number of base grid points.

The routine D03RYF can be called from INIDOM to obtain a simple graphical representation of the base grid, and to verify the data that you have specified in INIDOM.

Subgrids are stored internally using the same data structure, and solution information is communicated to you in PDEIV, PDEDEF and BNDARY in arrays according to the grid index on the particular level, e.g., X$(i)$ and Y$(i)$ contain the $(x, y)$ coordinates of grid point $i$, and U$(i, j)$ contains the $j$th solution component $u_j$ at grid point $i$.

The grid data and the solutions at all grid levels are 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 provided. The user-supplied monitor (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 argument TLAST. The routine D03RZF should be called from MONITR to obtain grid information at a particular level.

Further details of the underlying algorithm can be found in Section 9 and in Blom and Verwer (1993) and Blom *et al.* (1996) and the references therein.

## 4 References

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

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 Arguments

1: NPDE – INTEGER *Input*

*On entry*: the number of PDEs in the system.

*Constraint*: NPDE $\geq 1$.

2: TS – REAL (KIND=nag_wp) *Input/Output*

*On entry*: the initial value of the independent variable $t$.

*On exit*: the value of $t$ which has been reached. Normally TS = TOUT.

*Constraint*: TS $<$ TOUT.

3: TOUT – REAL (KIND=nag_wp) *Input*

*On entry*: the final value of $t$ to which the integration is to be carried out.

4: DT(3) – REAL (KIND=nag_wp) array *Input/Output*

*On entry*: 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 × (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 × ***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.

*On exit*: DT(1) contains the time step size for the next time step. DT(2) and DT(3) are unchanged or set to their default values if zero on entry.

*Constraints*:

if IND = 0, DT(1) $\geq 0.0$;
if IND = 0 and DT(1) $> 0.0$,
$10.0 \times$ ***machine precision*** $\times \max(|\text{TS}|, |\text{TOUT}|) \leq \text{DT}(1) \leq \text{TOUT} - \text{TS}$ and
$\text{DT}(2) \leq \text{DT}(1) \leq \text{DT}(3)$, where the values of DT(2) and DT(3) will have been reset to their default values if zero on entry;
$0 \leq \text{DT}(2) \leq \text{DT}(3)$.

5: TOLS – REAL (KIND=nag_wp) *Input*

*On entry*: the space tolerance used in the grid refinement strategy ($\sigma$ in equation (4)). See Section 9.2.

*Constraint*: TOLS $> 0.0$.

6: TOLT – REAL (KIND=nag_wp) *Input*

*On entry*: the time tolerance used to determine the time step size ($\tau$ in equation (7)). See Section 9.3.

*Constraint*: TOLT $> 0.0$.

7: INIDOM – SUBROUTINE, supplied by the user. *External Procedure*

INIDOM must specify the base grid in terms of the data structure described in Section 3. INIDOM is not referenced if, on entry, IND $= 1$. D03RYF can be called from INIDOM to obtain a simple graphical representation of the base grid, and to verify the data that you have specified in INIDOM. D03RBF also checks the validity of the data, but you are strongly advised to call D03RYF to ensure that the base grid is exactly as required.

**Note**: the boundaries of the base grid should consist of as many points as are necessary to employ second-order space discretization, i.e., a boundary enclosing the internal part of the domain must include at least 3 grid points including the corners. If Neumann boundary conditions are to be applied the minimum is 4.

The specification of INIDOM is:

```
SUBROUTINE INIDOM (MAXPTS, XMIN, XMAX, YMIN, YMAX, NX, NY, NPTS,     &
                   NROWS, NBNDS, NBPTS, LROW, IROW, ICOL, LLBND,     &
                   ILBND, LBND, IERR)

INTEGER            MAXPTS, NX, NY, NPTS, NROWS, NBNDS, NBPTS,        &
                   LROW(*), IROW(*), ICOL(*), LLBND(*),              &
                   ILBND(*), LBND(*), IERR
REAL (KIND=nag_wp) XMIN, XMAX, YMIN, YMAX
```

1: MAXPTS – INTEGER *Input*

*On entry*: the maximum number of base grid points allowed by the available workspace.

2: XMIN – REAL (KIND=nag_wp) *Output*

3: XMAX – REAL (KIND=nag_wp) *Output*

*On exit*: the extents of the virtual grid in the $x$-direction, i.e., the $x$ coordinates 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.

4: YMIN – REAL (KIND=nag_wp) *Output*

5: YMAX – REAL (KIND=nag_wp) *Output*

*On exit*: the extents of the virtual grid in the $y$-direction, i.e., the $y$ coordinates of the left and right boundaries respectively.

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

6: NX – INTEGER *Output*
7: NY – INTEGER *Output*

*On exit*: the number of virtual grid points in the $x$- and $y$-direction respectively (including the boundary points).

*Constraint*: NX and NY $\geq 4$.

8: NPTS – INTEGER *Output*

*On exit*: the total number of points in the base grid. If the required number of points is greater than MAXPTS then INIDOM must be exited immediately with IERR set to $-1$ to avoid overwriting memory.

*Constraint*: NPTS $\leq$ NX $\times$ NY and if IERR $\neq -1$ on exit, NPTS $\leq$ MAXPTS.

9: NROWS – INTEGER *Output*

*On exit*: the total number of rows of the virtual grid that contain base grid points. This is the maximum base row index.

*Constraint*: $4 \leq$ NROWS $\leq$ NY.

10: NBNDS – INTEGER *Output*

*On exit*: the total number of physical boundaries and corners in the base grid.

*Constraint*: NBNDS $\geq 8$.

11: NBPTS – INTEGER *Output*

*On exit*: the total number of boundary points in the base grid.

*Constraint*: $12 \leq$ NBPTS $<$ NPTS.

12: LROW$(*)$ – INTEGER array *Output*

*On exit*: LROW$(i)$, for $i = 1, 2, \ldots,$ NROWS, must contain the base grid index of the first grid point in base grid row $i$.

*Constraints*:

$1 \leq$ LROW$(i) \leq$ NPTS, for $i = 1, 2, \ldots,$ NROWS;
LROW$(i-1) <$ LROW$(i)$, for $i = 2, 3, \ldots,$ NROWS.

13: IROW$(*)$ – INTEGER array *Output*

*On exit*: IROW$(i)$, for $i = 1, 2, \ldots,$ NROWS, must contain the virtual row number $v_y$ that corresponds to base grid row $i$.

*Constraints*:

$0 \leq$ IROW$(i) \leq$ NY, for $i = 1, 2, \ldots,$ NROWS;
IROW$(i-1) <$ IROW$(i)$, for $i = 2, 3, \ldots,$ NROWS.

14: ICOL$(*)$ – INTEGER array *Output*

*On exit*: ICOL$(i)$, for $i = 1, 2, \ldots,$ NPTS, must contain the virtual column number $v_x$ that contains base grid point $i$.

*Constraint*: $0 \leq$ ICOL$(i) \leq$ NX, for $i = 1, 2, \ldots,$ NPTS.

15: LLBND$(*)$ – INTEGER array *Output*

*On exit*: LLBND$(i)$, for $i = 1, 2, \ldots,$ NBNDS, must contain the element of LBND corresponding to the start of the $i$th boundary or corner.

**Note**: the order of the boundaries and corners in LLBND must be first all the boundaries and then all the corners. The end points of a boundary (i.e., the adjacent corner points) must **not** be included in the list of points on that boundary. Also, if a corner is shared by two pairs of physical boundaries then it has two types and must therefore be treated as two corners.

*Constraints*:

$1 \leq \text{LLBND}(i) \leq \text{NBPTS}$, for $i = 1, 2, \ldots, \text{NBNDS}$;
$\text{LLBND}(i-1) < \text{LLBND}(i)$, for $i = 2, 3, \ldots, \text{NBNDS}$.

16: ILBND($*$) – INTEGER array *Output*

*On exit*: ILBND($i$), for $i = 1, 2, \ldots, \text{NBNDS}$, must contain the type of the $i$th boundary (or corner), as given in Section 3.

*Constraint*: ILBND($i$) must be equal to one of the following: 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for $i = 1, 2, \ldots, \text{NBNDS}$.

17: LBND($*$) – INTEGER array *Output*

*On exit*: LBND($i$), for $i = 1, 2, \ldots, \text{NBPTS}$, must contain the grid index of the $i$th boundary point. The order of the boundaries is as specified in LLBND, but within this restriction the order of the points in LBND is arbitrary.

*Constraint*: $1 \leq \text{LBND}(i) \leq \text{NPTS}$, for $i = 1, 2, \ldots, \text{NBPTS}$.

18: IERR – INTEGER *Input/Output*

*On entry*: will be initialized by D03RBF to some value prior to internal calls to INIDOM.

*On exit*: if the required number of grid points is larger than MAXPTS, IERR must be set to $-1$ to force a termination of the integration and an immediate return to the calling program with IFAIL $= 3$. Otherwise, IERR should remain unchanged.

INIDOM must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D03RBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

8: PDEDEF – SUBROUTINE, supplied by the user. *External Procedure*

PDEDEF must evaluate the functions $F_j$, for $j = 1, 2, \ldots, \text{NPDE}$, 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 BNDARY. PDEDEF is called for each subgrid in turn.

The specification of PDEDEF is:

```
SUBROUTINE PDEDEF (NPTS, NPDE, T, X, Y, U, UT, UX, UY, UXX, UXY,     &
                   UYY, RES)

INTEGER            NPTS, NPDE
REAL (KIND=nag_wp) T, X(NPTS), Y(NPTS), U(NPTS,NPDE),                &
                   UT(NPTS,NPDE), UX(NPTS,NPDE), UY(NPTS,NPDE),      &
                   UXX(NPTS,NPDE), UXY(NPTS,NPDE),                   &
                   UYY(NPTS,NPDE), RES(NPTS,NPDE)
```

1: NPTS – INTEGER *Input*

*On entry*: the number of grid points in the current grid.

2: NPDE – INTEGER *Input*

*On entry*: the number of PDEs in the system.

3: T – REAL (KIND=nag_wp) *Input*

*On entry*: the current value of the independent variable $t$.

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: X($i$) contains the $x$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS.

5: Y(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: Y($i$) contains the $y$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS.

6: U(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: U($i, j$) contains the value of the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

7: UT(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UT($i, j$) contains the value of $\dfrac{\partial u}{\partial t}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

8: UX(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UX($i, j$) contains the value of $\dfrac{\partial u}{\partial x}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

9: UY(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UY($i, j$) contains the value of $\dfrac{\partial u}{\partial y}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

10: UXX(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UXX($i, j$) contains the value of $\dfrac{\partial^2 u}{\partial x^2}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

11: UXY(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UXY($i, j$) contains the value of $\dfrac{\partial^2 u}{\partial x \partial y}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

12: UYY(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UYY($i, j$) contains the value of $\dfrac{\partial^2 u}{\partial y^2}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

13: RES(NPTS, NPDE) – REAL (KIND=nag_wp) array *Output*

*On exit*: RES($i, j$) must contain the value of $F_j$, for $j = 1, 2, \ldots,$ NPDE, at the $i$th 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.

PDEDEF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D03RBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

9: BNDARY – SUBROUTINE, supplied by the user. *External Procedure*

BNDARY must evaluate the functions $G_j$, for $j = 1, 2, \ldots, \text{NPDE}$, 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 subroutine.

The specification of BNDARY is:

```
SUBROUTINE BNDARY (NPTS, NPDE, T, X, Y, U, UT, UX, UY, NBNDS,        &
                   NBPTS, LLBND, ILBND, LBND, RES)
INTEGER            NPTS, NPDE, NBNDS, NBPTS, LLBND(NBNDS),           &
                   ILBND(NBNDS), LBND(NBPTS)
REAL (KIND=nag_wp) T, X(NPTS), Y(NPTS), U(NPTS,NPDE),                &
                   UT(NPTS,NPDE), UX(NPTS,NPDE), UY(NPTS,NPDE),      &
                   RES(NPTS,NPDE)
```

1: NPTS – INTEGER *Input*

*On entry*: the number of grid points in the current grid.

2: NPDE – INTEGER *Input*

*On entry*: the number of PDEs in the system.

3: T – REAL (KIND=nag_wp) *Input*

*On entry*: the current value of the independent variable $t$.

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: X($i$) contains the $x$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$.

5: Y(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: Y($i$) contains the $y$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$.

6: U(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: U($i, j$) contains the value of the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$ and $j = 1, 2, \ldots, \text{NPDE}$.

7: UT(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UT($i, j$) contains the value of $\dfrac{\partial u}{\partial t}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$ and $j = 1, 2, \ldots, \text{NPDE}$.

8: UX(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UX($i, j$) contains the value of $\dfrac{\partial u}{\partial x}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$ and $j = 1, 2, \ldots, \text{NPDE}$.

9: UY(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: UY($i, j$) contains the value of $\dfrac{\partial u}{\partial y}$ for the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots, \text{NPTS}$ and $j = 1, 2, \ldots, \text{NPDE}$.

10: NBNDS – INTEGER *Input*

*On entry*: the total number of physical boundaries and corners in the grid.

11: NBPTS – INTEGER *Input*

*On entry*: the total number of boundary points in the grid.

12: LLBND(NBNDS) – INTEGER array *Input*

*On entry*: LLBND($i$), for $i = 1, 2, \ldots,$ NBNDS, contains the element of LBND corresponding to the start of the $i$th boundary (or corner).

13: ILBND(NBNDS) – INTEGER array *Input*

*On entry*: ILBND($i$), for $i = 1, 2, \ldots,$ NBNDS, contains the type of the $i$th boundary, as given in Section 3.

14: LBND(NBPTS) – INTEGER array *Input*

*On entry*: LBND($i$), contains the grid index of the $i$th boundary point, where the order of the boundaries is as specified in LLBND. Hence the $i$th boundary point has coordinates X(LBND($i$)) and Y(LBND($i$)), and the corresponding solution values are U(LBND($i$), $j$), for $i = 1, 2, \ldots,$ NBPTS and $j = 1, 2, \ldots,$ NPDE.

15: RES(NPTS, NPDE) – REAL (KIND=nag_wp) array *Input/Output*

*On entry*: contains function values returned by PDEDEF.

*On exit*: RES(LBND($i$), $j$) must contain the value of $G_j$, for $j = 1, 2, \ldots,$ NPDE, at the $i$th boundary point, for $i = 1, 2, \ldots,$ NBPTS.

**Note**: elements of RES corresponding to interior points, i.e., points not included in LBND, must **not** be altered.

BNDARY must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03RBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

10: PDEIV – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of PDEIV is:

```
SUBROUTINE PDEIV (NPTS, NPDE, T, X, Y, U)
INTEGER            NPTS, NPDE
REAL (KIND=nag_wp) T, X(NPTS), Y(NPTS), U(NPTS,NPDE)
```

1: NPTS – INTEGER *Input*

*On entry*: the number of grid points in the base grid.

2: NPDE – INTEGER *Input*

*On entry*: the number of PDEs in the system.

3: T – REAL (KIND=nag_wp) *Input*

*On entry*: the (initial) value of the independent variable $t$.

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: X($i$) contains the $x$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS.

5: Y(NPTS) – REAL (KIND=nag_wp) array *Input*

*On entry*: Y($i$) contains the $y$ coordinate of the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS.

6: U(NPTS, NPDE) – REAL (KIND=nag_wp) array *Output*

*On exit*: U($i, j$) must contain the value of the $j$th PDE component at the $i$th grid point, for $i = 1, 2, \ldots,$ NPTS and $j = 1, 2, \ldots,$ NPDE.

PDEIV must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D03RBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

11: MONITR – SUBROUTINE, supplied by the user. *External Procedure*

MONITR is called by D03RBF 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 argument TLAST.

The input arguments contain information about the grid and solution at all grid levels used. D03RZF should be called from MONITR in order to extract the number of points and their $(x, y)$ coordinates on a particular grid.

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

The specification of MONITR is:

```
SUBROUTINE MONITR (NPDE, T, DT, DTNEW, TLAST, NLEV, XMIN, YMIN,      &
                   DXB, DYB, LGRID, ISTRUC, LSOL, SOL, IERR)
INTEGER            NPDE, NLEV, LGRID(*), ISTRUC(*), LSOL(NLEV),      &
                   IERR
REAL (KIND=nag_wp) T, DT, DTNEW, XMIN, YMIN, DXB, DYB, SOL(*)
LOGICAL            TLAST
```

1: NPDE – INTEGER *Input*

*On entry*: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

*On entry*: the current value of the independent variable $t$, i.e., the time at the end of the integration step just completed.

3: DT – REAL (KIND=nag_wp) *Input*

*On entry*: the current time step size $\Delta t$, i.e., the time step size used for the integration step just completed.

4: DTNEW – REAL (KIND=nag_wp) *Input*

*On entry*: the time step size that will be used for the next time step.

5: TLAST – LOGICAL *Input*

*On entry*: 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 – INTEGER *Input*

*On entry*: the number of grid levels used at time T.

7: XMIN – REAL (KIND=nag_wp) *Input*
8: YMIN – REAL (KIND=nag_wp) *Input*

*On entry*: the $(x, y)$ coordinates of the lower-left corner of the virtual grid.

9: DXB – REAL (KIND=nag_wp) *Input*
10: DYB – REAL (KIND=nag_wp) *Input*

*On entry*: the sizes of the base grid spacing in the $x$- and $y$-direction respectively.

11: LGRID$(*)$ – INTEGER array *Input*

*On entry*: contains pointers to the start of the grid structures in ISTRUC, and must be passed unchanged to D03RZF in order to extract the grid information.

12: ISTRUC$(*)$ – INTEGER array *Input*

*On entry*: contains the grid structures for each grid level and must be passed unchanged to D03RZF in order to extract the grid information.

13: LSOL(NLEV) – INTEGER array *Input*

*On entry*: 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.)

14: SOL$(*)$ – REAL (KIND=nag_wp) array *Input*

*On entry*: contains the solution $u$ at time T for each grid level $l$ in turn, positioned according to LSOL. More precisely

$$\mathrm{U}(i,j) = \mathrm{SOL}(\mathrm{LSOL}(l) + (j-1) \times n_l + i)$$

represents the $j$th component of the solution at the $i$th grid point in the $l$th level, for $i = 1, 2, \ldots, n_l$, $j = 1, 2, \ldots,$ NPDE and $l = 1, 2, \ldots,$ NLEV, where $n_l$ is the number of grid points at level $l$ (obtainable by a call to D03RZF).

15: IERR – INTEGER *Input/Output*

*On entry*: will be initialized by D03RBF to some value prior to internal calls to IERR.

*On exit*: should be set to 1 to force a termination of the integration and an immediate return to the calling program with IFAIL $= 4$. IERR should remain unchanged otherwise.

MONITR must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03RBF is called. Arguments denoted as *Input* must **not** be changed by this procedure.

12: OPTI(4) – INTEGER array *Input*

*On entry*: 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 $i = 2, 3, 4$, 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 $i = 2, 3, 4$.

13: OPTR(3, NPDE) – REAL (KIND=nag_wp) array *Input*

*On entry*: may be used to specify the optional vectors $u^{\max}$, $w^s$ and $w^t$ in the space and time monitors (see Section 9).

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 $j$th 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 $j$th 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 $j$th 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.

14: RWK(LENRWK) – REAL (KIND=nag_wp) array *Communication Array*

15: LENRWK – INTEGER *Input*

*On entry*: the dimension of the array RWK as declared in the (sub)program from which D03RBF is called.

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

$$\text{LENRWK} = maxpts \times \text{NPDE} \times (5 \times l + 18 \times \text{NPDE} + 9) + 2 \times maxpts,$$

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 routine returns with IFAIL $= 3$ and an estimated required size is printed on the current error message unit (see X04AAF).

**Note**: the size of LENRWK cannot be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

16: IWK(LENIWK) – INTEGER array *Communication Array*

*On entry*: if IND $= 0$, IWK need not be set. Otherwise IWK must remain unchanged from a previous call to D03RBF.

*On exit*: the following components of the array IWK concern the efficiency of the integration. Here, $m$ is the maximum number of grid levels allowed ($m =$ OPTI(1) if 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 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 D03RBF. 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.

17: LENIWK – INTEGER *Input*

*On entry*: the dimension of the array IWK as declared in the (sub)program from which D03RBF is called.

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

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

where $maxpts$ is the expected maximum number of grid points at any one level and $m = \text{OPTI}(1)$ if $\text{OPTI}(1) > 0$ and $m = 3$ otherwise. If during the execution the supplied value is found to be too small then the routine returns with IFAIL $= 3$ and an estimated required size is printed on the current error message unit (see X04AAF).

**Note**: the size of LENIWK cannot be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

18: LWK(LENLWK) – LOGICAL array *Workspace*

19: LENLWK – INTEGER *Input*

*On entry*: the dimension of the array LWK as declared in the (sub)program from which D03RBF is called.

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

$$\text{LENLWK} = maxpts + 1,$$

where $maxpts$ 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 routine returns with IFAIL $= 3$ and an estimated required size is printed on the current error message unit (see X04AAF).

**Note**: the size of LENLWK cannot be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

20: ITRACE – INTEGER *Input*

*On entry*: the level of trace information required from D03RBF. 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 X04ABF). 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 – INTEGER *Input/Output*

*On entry*: must be set to 0 or 1, alternatively 10 or 11.

IND $= 0$
Starts the integration in time. PDEDEF is assumed to be serial.

IND $= 1$
Continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RBF: TOUT, DT, TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL. PDEDEF is assumed to be serial.

IND $= 10$
Starts the integration in time. PDEDEF is assumed to have been parallelized by you, as described in Section 8. In all other respects, this is equivalent to IND $= 0$.

IND $= 11$
Continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RAF: TOUT, DT, TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL. PDEDEF is assumed to have been parallelized by you, as described in Section 8. In all other respects, this is equivalent to IND $= 1$.

*Constraint*: $0 \leq \text{IND} \leq 1$ or $10 \leq \text{IND} \leq 11$.

*On exit*: IND $= 1$, if IND on input was 0 or 1, or IND $= 11$, if IND on input was 10 or 11.

**Note**: for users of serial versions of the NAG Library, it is recommended that you only use IND $= 0$ or 1. See Section 8 for more information on the use of IND.

22: IFAIL – INTEGER *Input/Output*

*On entry*: IFAIL must be set to 0, $-1$ or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value $-1$ or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value $-1$ or 1 is used it is essential to test the value of IFAIL on exit.**

*On exit*: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

## 6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or $-1$, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

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 $= 0$ and $0.0 <$ DT$(1) < 10 \times$ ***machine precision*** $\times \max(|\text{TS}|, |\text{TOUT}|)$,
or IND $= 0$ and DT$(1) >$ TOUT $-$ TS,
or IND $= 0$ and DT$(1) <$ DT$(2)$ or DT$(1) >$ DT$(3)$,
or TOLS or TOLT $\leq 0.0$,
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 IND $\neq 0$ or 1,
or IND $= 1$ on initial entry to D03RBF.

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 PDEDEF and/or BNDARY, and the initial values in PDEIV if appropriate.

IFAIL $= 3$

One or more of the workspace arrays is too small for the required number of grid points. At the initial time step this error may result because you set IERR to $-1$ in INIDOM or the internal check on the number of grid points following the call to INIDOM. 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 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.

IFAIL $= 6$

One or more of the output arguments of INIDOM was incorrectly specified, i.e.,

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 NROWS $< 4$,
or NROWS $>$ NY,
or NPTS $>$ NX $\times$ NY,
or NBNDS $< 8$,
or NBPTS $< 12$,
or NBPTS $\geq$ NPTS,
or LROW$(i) < 1$ or LROW$(i) >$ NPTS, for some $i = 1, 2, \ldots,$ NROWS,
or LROW$(i) \leq$ LROW$(i-1)$, for some $i = 2, 3, \ldots,$ NROWS,
or IROW$(i) < 0$ or IROW$(i) >$ NY, for some $i = 1, 2, \ldots,$ NROWS,
or IROW$(i) \leq$ IROW$(i-1)$, for some $i = 2, 3, \ldots,$ NROWS,
or ICOL$(i) < 0$ or ICOL$(i) >$ NX, for some $i = 1, 2, \ldots,$ NPTS,
or LLBND$(i) < 1$ or LLBND$(i) >$ NBPTS, for some $i = 1, 2, \ldots,$ NBNDS,
or LLBND$(i) \leq$ LLBND$(i-1)$, for some $i = 2, 3, \ldots,$ NBNDS,
or ILBND$(i) \neq$ 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for some $i = 1, 2, \ldots,$ NBNDS,
or LBND$(i) < 1$ or LBND$(i) >$ NPTS, for some $i = 1, 2, \ldots,$ NBPTS.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

## 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 arguments TOLS and TOLT described in Section 9, and you should test the effects of varying these arguments. 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 Parallelism and Performance

D03RBF requires a user-supplied routine PDEDEF to evaluate the functions $F_j$, for $j = 1, 2, \ldots,$ NPDE. The parallelism within D03RBF will be more efficient if PDEDEF can also be parallelized. This is often the case, but you must add some OpenMP directives to your version of PDEDEF to implement the parallelism. For example, if the body of code for PDEDEF is as follows (adapted from the first test case in the document for D03RBF):

```
res(1:npts,1:npde) = ut(1:npts,1:npde) - diffusion*(uxx(1:npts,1:          &
   npde)+uyy(1:npts,1:npde)) - damkohler*(one+heat_release-u(1:npts,       &
    1:npde))*exp(-activ_energy/u(1:npts,1:npde))
```

This example can be parallelized, as the updating of RES for each value in the range $1, \ldots,$ NPTS is independent of every other value. Thus this should be parallelized in OpenMP (using an explicit loop rather than Fortran array syntax) as follows:

```
!$OMP DO
    Do i = 1, npts
       res(i,1:npde) = ut(i,1:npde) -diffusion*(uxx(i,1:npde)+uyy(i,1:npde &
        )) - damkohler*(1.0E0_nag_wp+heat_release-u(i,1:npde))*exp(-       &
         activ_energy/u(i,1:npde))
    End Do
 !$OMP END DO
```

Note that the OpenMP PARALLEL directive must **not** be specified, as the OpenMP DO directive will bind to the PARALLEL region within the D03RBF code. Also note that this assumes the default OpenMP behaviour that all variables are SHARED, except for loop indices that are PRIVATE.

To avoid problems for existing library users, who will not have specified any OpenMP directives in their PDEDEF routine, the default assumption of D03RBF is that PDEDEF has not been parallelized, and executes calls to PDEDEF in serial mode. You must indicate that PDEDEF has been parallelized by setting IND to 10 or 11 as appropriate. See Section 5 for details.

If the code within PDEDEF cannot be parallelized, you must **not** add any OpenMP directives to your code, and must **not** set IND to 10 or 11. If IND is set to 10 or 11 and PDEDEF has not been parallelized, results on multiple threads will be unpredictable and may give rise to incorrect results and/or program crashes or deadlocks. Please contact NAG for advice if required. Overloading IND in this manner is not entirely satisfactory, consequently it is likely that replacement interfaces for D03RBF will be included in a future NAG Library release.

# 9 Further Comments

## 9.1 Algorithm Outline

The local uniform grid refinement method is summarised 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.

## 9.2 Refinement Strategy

For each grid point $i$ a space monitor $\mu_i^s$ is determined by

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

where $\Delta x$ and $\Delta y$ are the grid widths in the $x$ and $y$ directions; and $x_i$, $y_i$ are the $(x, y)$ coordinates at grid point $i$. The argument $\gamma_j$ is obtained from

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

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

A new level of refinement is created if

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

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 9.1.

### 9.3 Time Integration

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

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

where $ngpts(l)$ is the total number of points on grid level $l$; $N = ngpts(l) \times \text{NPDE}$; $\Delta t$ 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 $\alpha_{ij}$ 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 $\tau$ is the user-specified time tolerance.

An integration step is rejected and retried at all levels if

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

## 10 Example

This example is taken from Blom and Verwer (1993) and is the two-dimensional Burgers' system

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} + \epsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$
$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} + \epsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),$$

with $\epsilon = 10^{-3}$ on the domain given in Figure 3. Dirichlet boundary conditions are used on all boundaries using the exact solution

$$u = \tfrac{3}{4} - \frac{1}{4(1 + \exp((-4x + 4y - t)/(32\epsilon)))},$$
$$v = \tfrac{3}{4} + \frac{1}{4(1 + \exp((-4x + 4y - t)/(32\epsilon)))}.$$

The solution contains a wave front at $y = x + 0.25t$ which propagates in a direction perpendicular to the front with speed $\sqrt{2}/8$.

## 10.1 Program Text

```
!   D03RBF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d03rbfe_mod

!     D03RBF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                            :: bndary, inidom, monitr, pdedef,      &
                                           pdeiv
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter, Public :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: twant(2) = (/0.25_nag_wp,one/)
      Integer, Parameter, Public        :: itrace = 0, nin = 5, nout = 6,       &
                                           npde = 2
!     .. Local Scalars ..
      Integer, Public, Save             :: iout
    Contains
      Subroutine pdeiv(npts,npde,t,x,y,u)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: eps = 0.001_nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)           :: npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: u(npts,npde)
        Real (Kind=nag_wp), Intent (In) :: x(npts), y(npts)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: a
        Integer                        :: i
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        Do i = 1, npts
          a = (4.0_nag_wp*(y(i)-x(i))-t)/(32.0_nag_wp*eps)
          If (a<=zero) Then
            u(i,1) = 0.75_nag_wp - 0.25_nag_wp/(one+exp(a))
            u(i,2) = 0.75_nag_wp + 0.25_nag_wp/(one+exp(a))
          Else
            a = -a
            u(i,1) = 0.75_nag_wp - 0.25_nag_wp*exp(a)/(one+exp(a))
            u(i,2) = 0.75_nag_wp + 0.25_nag_wp*exp(a)/(one+exp(a))
          End If
        End Do

        Return
      End Subroutine pdeiv
      Subroutine inidom(maxpts,xmin,xmax,ymin,ymax,nx,ny,npts,nrows,nbnds,     &
        nbpts,lrow,irow,icol,llbnd,ilbnd,lbnd,ierr)

!       .. Use Statements ..
        Use nag_library, Only: d03ryf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: xmax, xmin, ymax, ymin
        Integer, Intent (Inout)        :: ierr
        Integer, Intent (In)           :: maxpts
```

```
      Integer, Intent (Out)          :: nbnds, nbpts, npts, nrows, nx, ny
!     .. Array Arguments ..
      Integer, Intent (Inout)        :: icol(*), ilbnd(*), irow(*), lbnd(*), &
                                        llbnd(*), lrow(*)
!     .. Local Scalars ..
      Integer                        :: i, ifail, j, leniwk
!     .. Local Arrays ..
      Integer                        :: icold(105), ilbndd(28), irowd(11),   &
                                        iwk(122), lbndd(72), llbndd(28),     &
                                        lrowd(11)
      Character (33)                 :: pgrid(11)
!     .. Executable Statements ..
      icold(1:105) = (/0,1,2,0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,9,10,  &
        0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,8,9,10,0,1,2,3,4,5,6,7,8,9,10,0,  &
        1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,9,10,0,1,2,3,4,5,6,7,8,0,1,2, &
        3,4,5,6,7,8,0,1,2,3,4,5,6,7,8/)
      ilbndd(1:28) = (/1,2,3,4,1,4,1,2,3,4,3,4,1,2,12,23,34,41,14,41,12,23,  &
        34,41,43,14,21,32/)
      irowd(1:11) = (/0,1,2,3,4,5,6,7,8,9,10/)

      lbndd(1:72) = (/2,4,15,26,37,46,57,68,79,88,98,99,100,101,102,103,104, &
        96,86,85,84,83,82,70,59,48,39,28,17,6,8,9,10,11,12,13,18,29,40,49,   &
        60,72,73,74,75,76,77,67,56,45,36,25,33,32,42,52,53,43,1,97,105,87,   &
        81,3,7,71,78,14,31,51,54,34/)

      llbndd(1:28) = (/1,2,11,18,19,24,31,37,42,48,53,55,56,58,59,60,61,62,  &
        63,64,65,66,67,68,69,70,71,72/)
      lrowd(1:11) = (/1,4,15,26,37,46,57,68,79,88,97/)

      nx = 11
      ny = 11

!     Check MAXPTS against rough estimate of NPTS

      npts = nx*ny
      If (maxpts<npts) Then
        ierr = -1
        Return
      End If

      xmin = zero
      ymin = zero
      xmax = one
      ymax = one

      nrows = 11
      npts = 105
      nbnds = 28
      nbpts = 72

      Do i = 1, nrows
        lrow(i) = lrowd(i)
        irow(i) = irowd(i)
      End Do

      Do i = 1, nbnds
        llbnd(i) = llbndd(i)
        ilbnd(i) = ilbndd(i)
      End Do

      Do i = 1, nbpts
        lbnd(i) = lbndd(i)
      End Do

      Do i = 1, npts
        icol(i) = icold(i)
      End Do

      Write (nout,*) 'Base grid:'
      Write (nout,*)
      leniwk = 122
```

```
        ifail = -1

        Call d03ryf(nx,ny,npts,nrows,nbnds,nbpts,lrow,irow,icol,llbnd,ilbnd,   &
          lbnd,iwk,leniwk,pgrid,ifail)

        If (ifail==0) Then
          Write (nout,*) ' '
          Do j = 1, ny
            Write (nout,*) pgrid(j)
            Write (nout,*) ' '
          End Do
          Write (nout,*) ' '
        End If

        Return
      End Subroutine inidom
      Subroutine pdedef(npts,npde,t,x,y,u,ut,ux,uy,uxx,uxy,uyy,res)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: eps = 1E-3_nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)           :: npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: res(npts,npde)
        Real (Kind=nag_wp), Intent (In) :: u(npts,npde), ut(npts,npde),        &
                                           ux(npts,npde), uxx(npts,npde),      &
                                           uxy(npts,npde), uy(npts,npde),      &
                                           uyy(npts,npde), x(npts), y(npts)
!       .. Local Scalars ..
        Integer                        :: i
!       .. Executable Statements ..
        !$Omp Do
        Do i = 1, npts
          res(i,1) = -u(i,1)*ux(i,1) - u(i,2)*uy(i,1) +                        &
            eps*(uxx(i,1)+uyy(i,1))
          res(i,2) = -u(i,1)*ux(i,2) - u(i,2)*uy(i,2) +                        &
            eps*(uxx(i,2)+uyy(i,2))
          res(i,1) = ut(i,1) - res(i,1)
          res(i,2) = ut(i,2) - res(i,2)
        End Do
        !$Omp End Do

        Return
      End Subroutine pdedef
      Subroutine bndary(npts,npde,t,x,y,u,ut,ux,uy,nbnds,nbpts,llbnd,ilbnd,    &
        lbnd,res)

!       .. Parameters ..
        Real (Kind=nag_wp), Parameter  :: eps = 1E-3_nag_wp
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)           :: nbnds, nbpts, npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Inout) :: res(npts,npde)
        Real (Kind=nag_wp), Intent (In) :: u(npts,npde), ut(npts,npde),        &
                                           ux(npts,npde), uy(npts,npde),       &
                                           x(npts), y(npts)
        Integer, Intent (In)           :: ilbnd(nbnds), lbnd(nbpts),           &
                                           llbnd(nbnds)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: a
        Integer                        :: i, k
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        Do k = llbnd(1), nbpts
          i = lbnd(k)
          a = (-4.0_nag_wp*x(i)+4.0_nag_wp*y(i)-t)/(32.0_nag_wp*eps)
          If (a<=zero) Then
            res(i,1) = 0.75_nag_wp - 0.25_nag_wp/(one+exp(a))
```

```
            res(i,2) = 0.75_nag_wp + 0.25_nag_wp/(one+exp(a))
          Else
            a = -a
            res(i,1) = 0.75_nag_wp - 0.25_nag_wp*exp(a)/(one+exp(a))
            res(i,2) = 0.75_nag_wp + 0.25_nag_wp*exp(a)/(one+exp(a))
          End If
          res(i,1:2) = u(i,1:2) - res(i,1:2)
        End Do

        Return
      End Subroutine bndary
      Subroutine monitr(npde,t,dt,dtnew,tlast,nlev,xmin,ymin,dxb,dyb,lgrid,    &
        istruc,lsol,sol,ierr)

!       .. Use Statements ..
        Use nag_library, Only: d03rzf
!       .. Parameters ..
        Integer, Parameter             :: maxpts = 2500, nout = 6
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: dt, dtnew, dxb, dyb, t, xmin, ymin
        Integer, Intent (Inout)        :: ierr
        Integer, Intent (In)           :: nlev, npde
        Logical, Intent (In)           :: tlast
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In) :: sol(*)
        Integer, Intent (In)           :: istruc(*), lgrid(*), lsol(nlev)
!       .. Local Scalars ..
        Integer                        :: i, ifail, ipsol, level, npts
!       .. Local Arrays ..
        Real (Kind=nag_wp)             :: uex(105,2), x(maxpts), y(maxpts)
!       .. Executable Statements ..
        ifail = -1
levels: Do level = 1, nlev
          If (.Not. tlast) Then
            Exit levels
          End If
          ipsol = lsol(level)

!         Get grid information

          Call d03rzf(level,nlev,xmin,ymin,dxb,dyb,lgrid,istruc,npts,x,y,      &
            maxpts,ifail)
          If (ifail/=0) Then
            ierr = 1
            Exit levels
          End If

!         Skip printing if iout<2 or level>1.
          If (iout/=2 .Or. level/=1) Then
            Cycle levels
          End If

!         Get exact solution
          Call pdeiv(npts,npde,t,x,y,uex)
          Write (nout,*)
          Write (nout,99999) t
          Write (nout,*)
          Write (nout,99998) 'x', 'y', 'approx u', 'exact u', 'approx v',      &
            'exact v'
          Write (nout,*)
          ipsol = lsol(level)
          Do i = 1, npts, 2
            Write (nout,99997) x(i), y(i), sol(ipsol+i), uex(i,1),             &
              sol(ipsol+npts+i), uex(i,2)
          End Do
          Write (nout,*)

        End Do levels

        Return
99999   Format (' Solution at every 2nd grid point in level 1 at time ',F8.4,  &
```

```
          ':')
99998   Format (7X,A,9X,A,6X,A,2X,A,2X,A,2X,A)
99997   Format (6(1X,F9.2))
      End Subroutine monitr
    End Module d03rbfe_mod
    Program d03rbfe

!     D03RBF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d03rbf, nag_wp
      Use d03rbfe_mod, Only: bndary, inidom, iout, itrace, monitr, nin, nout,  &
                             npde, one, pdedef, pdeiv, twant, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: tols, tolt, tout, ts
      Integer                         :: i, ifail, ind, j, leniwk, lenlwk,    &
                                         lenrwk, maxlev, mxlev, npts
!     .. Local Arrays ..
      Real (Kind=nag_wp)              :: dt(3)
      Real (Kind=nag_wp), Allocatable :: optr(:,:), rwk(:)
      Integer, Allocatable            :: iwk(:)
      Integer                         :: opti(4)
      Logical, Allocatable            :: lwk(:)
!     .. Executable Statements ..
      Write (nout,*) 'D03RBF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) npts, mxlev

      leniwk = npts*(5*mxlev+14) + 2 + 7*mxlev
      lenlwk = 2*npts
      lenrwk = npts*npde*(5*mxlev+9+18*npde) + 2*npts
      Allocate (rwk(lenrwk),iwk(leniwk),lwk(lenlwk),optr(3,npde))

!     Specify that we are starting the integration in time (ind = 0 normally).
!     Note: we have parallelized the loop in the function pdedef using OpenMP
!     so set the alternative value of ind to indicate that this can be run in
!     parallel if we are using a multithreaded implementation. Either option
!     is OK for serial NAG Library implementations from Mark 25 onwards.
      ind = 10

      ts = zero
      dt(1) = 0.001_nag_wp
      dt(2) = 1.0E-7_nag_wp
      dt(3) = zero
      tols = 0.1_nag_wp
      tolt = 0.05_nag_wp
      opti(1) = 5
      maxlev = opti(1)
      opti(2:4) = 0
      optr(1:3,1:npde) = one

!     Call main routine
      Do iout = 1, 2
        tout = twant(iout)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d03rbf(npde,ts,tout,dt,tols,tolt,inidom,pdedef,bndary,pdeiv,      &
          monitr,opti,optr,rwk,lenrwk,iwk,leniwk,lwk,lenlwk,itrace,ind,ifail)

!       Print statistics

        Write (nout,99999) 'Statistics:'
        Write (nout,99998) 'Time = ', ts
        Write (nout,99997) 'Total number of accepted timesteps =', iwk(1)
        Write (nout,99997) 'Total number of rejected timesteps =', iwk(2)
        Write (nout,'(1X,4(/,A,A))') '            Total  number  of     ',     &
```

```
          '  maximum  number  of     ',                                        &
          '       Residual Jacobian    Newton   ', '      Newton   ',          &
          '        evals     evals     iters   ', '      iters   ', ' Level '

        maxlev = opti(1)
        Do j = 1, maxlev
          If (iwk(j+2)/=0) Then
            Write (nout,'(I4,4I10)') j, (iwk(j+2+i*maxlev),i=0,2),             &
              iwk(j+2+4*maxlev)
          End If
        End Do
        Write (nout,*)

      End Do

99999 Format (1X,A)
99998 Format (1X,A,F8.4)
99997 Format (1X,A,I5)
    End Program d03rbfe
```

## 10.2 Program Data

```
D03RBF Example Program Data
 3000  5           : npts, mxlev
```

## 10.3 Program Results

```
 D03RBF Example Program Results
 Base grid:


  23  3  3  3  3  3  3  3 34 XX XX

   2 .. .. .. .. .. .. ..  4 XX XX

   2 .. 14  1  1  1  1  1 41 XX XX

   2 ..  4 23  3  3  3  3  3  3 34

   2 ..  4  2 .. .. .. .. .. ..  4

   2 ..  4  2 .. 14  1  1 21 ..  4

   2 ..  4  2 ..  4 XX XX  2 ..  4

   2 ..  4  2 .. 43  3  3 32 ..  4

   2 ..  4  2 .. .. .. .. .. ..  4

   2 ..  4 12  1  1  1  1  1  1 41

  12  1 41 XX XX XX XX XX XX XX XX


 Statistics:
 Time =   0.2500
 Total number of accepted timesteps =   14
 Total number of rejected timesteps =    0

           Total  number  of       maximum  number  of
      Residual Jacobian    Newton         Newton
       evals     evals     iters          iters
 Level
   1       196        14        28         2
   2       196        14        28         2
   3       196        14        28         2
   4       196        14        28         2
   5       141        10        21         3
```

```
 Solution at every 2nd grid point in level 1 at time   1.0000:

      x         y      approx u  exact u  approx v  exact v

     0.00      0.00      0.50      0.50      1.00      1.00
     0.20      0.00      0.50      0.50      1.00      1.00
     0.10      0.10      0.50      0.50      1.00      1.00
     0.30      0.10      0.50      0.50      1.00      1.00
     0.50      0.10      0.50      0.50      1.00      1.00
     0.70      0.10      0.50      0.50      1.00      1.00
     0.90      0.10      0.50      0.50      1.00      1.00
     0.00      0.20      0.50      0.50      1.00      1.00
     0.20      0.20      0.50      0.50      1.00      1.00
     0.40      0.20      0.50      0.50      1.00      1.00
     0.60      0.20      0.50      0.50      1.00      1.00
     0.80      0.20      0.50      0.50      1.00      1.00
     1.00      0.20      0.50      0.50      1.00      1.00
     0.10      0.30      0.50      0.50      1.00      1.00
     0.30      0.30      0.50      0.50      1.00      1.00
     0.50      0.30      0.50      0.50      1.00      1.00
     0.70      0.30      0.50      0.50      1.00      1.00
     0.90      0.30      0.50      0.50      1.00      1.00
     0.00      0.40      0.75      0.75      0.75      0.75
     0.20      0.40      0.50      0.50      1.00      1.00
     0.40      0.40      0.50      0.50      1.00      1.00
     0.80      0.40      0.50      0.50      1.00      1.00
     1.00      0.40      0.50      0.50      1.00      1.00
     0.10      0.50      0.75      0.75      0.75      0.75
     0.30      0.50      0.50      0.50      1.00      1.00
     0.50      0.50      0.50      0.50      1.00      1.00
     0.70      0.50      0.50      0.50      1.00      1.00
     0.90      0.50      0.50      0.50      1.00      1.00
     0.00      0.60      0.75      0.75      0.75      0.75
     0.20      0.60      0.75      0.75      0.75      0.75
     0.40      0.60      0.50      0.50      1.00      1.00
     0.60      0.60      0.50      0.50      1.00      1.00
     0.80      0.60      0.50      0.50      1.00      1.00
     1.00      0.60      0.50      0.50      1.00      1.00
     0.10      0.70      0.75      0.75      0.75      0.75
     0.30      0.70      0.75      0.75      0.75      0.75
     0.50      0.70      0.50      0.50      1.00      1.00
     0.70      0.70      0.50      0.50      1.00      1.00
     0.90      0.70      0.50      0.50      1.00      1.00
     0.00      0.80      0.75      0.75      0.75      0.75
     0.20      0.80      0.75      0.75      0.75      0.75
     0.40      0.80      0.75      0.75      0.75      0.75
     0.60      0.80      0.50      0.50      1.00      1.00
     0.80      0.80      0.50      0.50      1.00      1.00
     0.10      0.90      0.75      0.75      0.75      0.75
     0.30      0.90      0.75      0.75      0.75      0.75
     0.50      0.90      0.75      0.75      0.75      0.75
     0.70      0.90      0.50      0.50      1.00      1.00
     0.00      1.00      0.75      0.75      0.75      0.75
     0.20      1.00      0.75      0.75      0.75      0.75
     0.40      1.00      0.75      0.75      0.75      0.75
     0.60      1.00      0.75      0.75      0.75      0.75
     0.80      1.00      0.50      0.50      1.00      1.00

 Statistics:
 Time =   1.0000
 Total number of accepted timesteps =   45
 Total number of rejected timesteps =    0

           Total  number  of       maximum  number  of
      Residual Jacobian    Newton          Newton
       evals     evals     iters           iters
 Level
   1       630        45        90          2
```

| | | | | |
|---|---|---|---|---|
| 2 | 630 | 45 | 90 | 2 |
| 3 | 630 | 45 | 90 | 2 |
| 4 | 630 | 45 | 90 | 2 |
| 5 | 575 | 41 | 83 | 3 |