D03EEF – NAG Fortran Library Routine Document

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

D03EEF discretizes a second-order elliptic partial differential equation (PDE) on a rectangular region.

2 Specification

```
      SUBROUTINE D03EEF(XMIN, XMAX, YMIN, YMAX, PDEF, BNDY, NGX, NGY,
     1                  LDA, A, RHS, SCHEME, IFAIL)
      INTEGER           NGX, NGY, LDA, IFAIL
      real              XMIN, XMAX, YMIN, YMAX, A(LDA,7), RHS(LDA)
      CHARACTER*1       SCHEME
      EXTERNAL          PDEF, BNDY
```

3 Description

D03EEF discretizes a second-order linear elliptic partial differential equation of the form

$$\alpha(x,y)\frac{\partial^2 U}{\partial x^2}+\beta(x,y)\frac{\partial^2 U}{\partial x\partial y}+\gamma(x,y)\frac{\partial^2 U}{\partial y^2}+\delta(x,y)\frac{\partial U}{\partial x}+\epsilon(x,y)\frac{\partial U}{\partial y}+\phi(x,y)U=\psi(x,y) \qquad (1)$$

on a rectangular region

$$x_A \le x \le x_B$$
$$y_A \le y \le y_B$$

subject to boundary conditions of the form

$$a(x,y)U+b(x,y)\frac{\partial U}{\partial n}=c(x,y)$$

where $\frac{\partial U}{\partial n}$ denotes the outward pointing normal derivative on the boundary. Equation (1) is said to be elliptic if

$$4\alpha(x,y)\gamma(x,y)\ge(\beta(x,y))^2$$

for all points in the rectangular region. The linear equations produced are in a form suitable for passing directly to the multigrid routine D03EDF.

The equation is discretized on a rectangular grid, with n_x grid points in the x-direction and n_y grid points in the y-direction. The grid spacing used is therefore

$$h_x=(x_B-x_A)/(n_x-1)$$
$$h_y=(y_B-y_A)/(n_y-1)$$

and the co-ordinates of the grid points (x_i,y_j) are

$$x_i=x_A+(i-1)h_x,\quad i=1,2,\ldots,n_x,$$
$$y_j=y_A+(j-1)h_y,\quad j=1,2,\ldots,n_y.$$

At each grid point (x_i,y_j) six neighbouring grid points are used to approximate the partial differential equation, so that the equation is discretized on the seven-point stencil shown in Figure 1.

For convenience the approximation u_{ij} to the exact solution $U(x_i,y_j)$ is denoted by u_{O}, and the neighbouring approximations are labelled according to points of the compass as shown. Where numerical labels for the seven points are required, these are also shown.

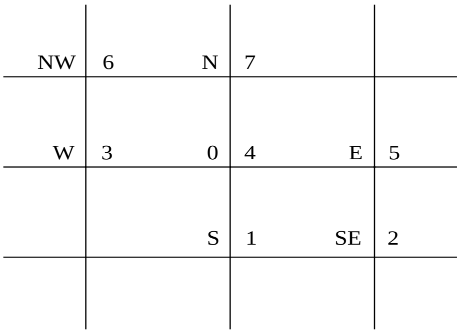

Figure 1

The following approximations are used for the second derivatives:

$$\frac{\partial^2 U}{\partial x^2} \simeq \frac{1}{h_x^2}(u_{\mathrm{E}} - 2u_{\mathrm{O}} + u_{\mathrm{W}})$$

$$\frac{\partial^2 U}{\partial y^2} \simeq \frac{1}{h_y^2}(u_{\mathrm{N}} - 2u_{\mathrm{O}} + u_{\mathrm{S}})$$

$$\frac{\partial^2 U}{\partial x \partial y} \simeq \frac{1}{2h_x h_y}(u_{\mathrm{N}} - u_{\mathrm{NW}} + u_{\mathrm{E}} - 2u_{\mathrm{O}} + u_{\mathrm{W}} - u_{\mathrm{SE}} + u_{\mathrm{S}}).$$

Two possible schemes may be used to approximate the first derivatives:

Central Differences

$$\frac{\partial U}{\partial x} \simeq \frac{1}{2h_x}(u_{\mathrm{E}} - u_{\mathrm{W}})$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{2h_y}(u_{\mathrm{N}} - u_{\mathrm{S}})$$

Upwind Differences

$$\frac{\partial U}{\partial x} \simeq \frac{1}{h_x}(u_{\mathrm{O}} - u_{\mathrm{W}}) \quad \text{if} \quad \delta(x,y) > 0$$

$$\frac{\partial U}{\partial x} \simeq \frac{1}{h_x}(u_{\mathrm{E}} - u_{\mathrm{O}}) \quad \text{if} \quad \delta(x,y) < 0$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{h_y}(u_{\mathrm{N}} - u_{\mathrm{O}}) \quad \text{if} \quad \epsilon(x,y) > 0$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{h_y}(u_{\mathrm{O}} - u_{\mathrm{S}}) \quad \text{if} \quad \epsilon(x,y) < 0.$$

Central differences are more accurate than upwind differences, but upwind differences may lead to a more diagonally dominant matrix for those problems where the coefficients of the first derivatives are significantly larger than the coefficients of the second derivatives.

The approximations used for the first derivatives may be written in a more compact form as follows:

$$\frac{\partial U}{\partial x} \simeq \frac{1}{2h_x}\left((k_x - 1)u_{\mathrm{W}} - 2k_x u_{\mathrm{O}} + (k_x + 1)u_{\mathrm{E}}\right)$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{2h_y}\left((k_y - 1)u_{\mathrm{S}} - 2k_y u_{\mathrm{O}} + (k_y + 1)u_{\mathrm{N}}\right)$$

where $k_x = \operatorname{sign}\delta$ and $k_y = \operatorname{sign}\epsilon$ for upwind differences, and $k_x = k_y = 0$ for central differences.

At all points in the rectangular domain, including the boundary, the coefficients in the partial differential equation are evaluated by calling the user-supplied subroutine PDEF, and applying the approximations.

This leads to a seven-diagonal system of linear equations of the form:

$$\begin{array}{llll} & A^6_{ij}u_{i-1,j+1} & + \; A^7_{ij}u_{i,j+1} & \\ + & A^3_{ij}u_{i-1,j} & + \; A^4_{ij}u_{ij} & + \; A^5_{ij}u_{i+1,j} \\ & & + \; A^1_{ij}u_{i,j-1} & + \; A^2_{ij}u_{i+1,j-1} = f_{ij}, \; i = 1,2,\ldots,n_x; j = 1,2,\ldots,n_y, \end{array}$$

where the coefficients are given by

$$\begin{aligned}
A^1_{ij} &= \beta(x_i,y_j)\frac{1}{2h_xh_y} + \gamma(x_i,y_j)\frac{1}{h_y^2} + \epsilon(x_i,y_j)\frac{1}{2h_y}(k_y - 1) \\
A^2_{ij} &= -\beta(x_i,y_j)\frac{1}{2h_xh_y} \\
A^3_{ij} &= \alpha(x_i,y_j)\frac{1}{h_x^2} + \beta(x_i,y_j)\frac{1}{2h_xh_y} + \delta(x_i,y_j)\frac{1}{2h_x}(k_x - 1) \\
A^4_{ij} &= -\alpha(x_i,y_j)\frac{2}{h_x^2} - \beta(x_i,y_j)\frac{1}{h_xh_y} - \gamma(x_i,y_j)\frac{2}{h_y^2} - \delta(x_i,y_j)\frac{k_y}{h_x} - \epsilon(x_i,y_j)\frac{k_y}{h_y} - \phi(x_i,y_j) \\
A^5_{ij} &= \alpha(x_i,y_j)\frac{1}{h_x^2} + \beta(x_i,y_j)\frac{1}{2h_xh_y} + \delta(x_i,y_j)\frac{1}{2h_x}(k_x + 1) \\
A^6_{ij} &= -\beta(x_i,y_j)\frac{1}{2h_xh_y} \\
A^7_{ij} &= \beta(x_i,y_j)\frac{1}{2h_xh_y} + \gamma(x_i,y_j)\frac{1}{h_y^2} + \epsilon(x_i,y_j)\frac{1}{2h_y}(k_y + 1) \\
f_{ij} &= \psi(x_i,y_j)
\end{aligned}$$

These equations then have to be modified to take account of the boundary conditions. These may be Dirichlet (where the solution is given), Neumann (where the derivative of the solution is given), or mixed (where a linear combination of solution and derivative is given).

If the boundary conditions are Dirichlet, there are an infinity of possible equations which may be applied:

$$\mu u_{ij} = \mu f_{ij} \; , \; \mu \neq 0. \qquad (2)$$

If D03EDF is used to solve the discretized equations, it turns out that the choice of μ can have a dramatic effect on the rate of convergence, and the obvious choice $\mu = 1$ is not the best. Some choices may even cause the multigrid method to fail altogether. In practice it has been found that a value of the same order as the other diagonal elements of the matrix is best, and the following value has been found to work well in practice:

$$\mu = \min_{ij}\left(-\left\{\frac{2}{h_x^2} + \frac{2}{h_y^2}\right\}, A^4_{ij}\right).$$

If the boundary conditions are either mixed or Neumann (i.e., B $\neq 0$ on return from the user-supplied subroutine BNDY), then one of the points in the seven-point stencil lies outside the domain. In this case the normal derivative in the boundary conditions is used to eliminate the 'fictitious' point, u_{outside}:

$$\frac{\partial U}{\partial n} \simeq \frac{1}{2h}(u_{\text{outside}} - u_{\text{inside}}). \qquad (3)$$

It should be noted that if the boundary conditions are Neumann and $\phi(x,y) \equiv 0$, then there is no unique solution. The routine returns with IFAIL = 5 in this case, and the seven-diagonal matrix is singular.

The four corners are treated separately. The user-supplied subroutine BNDY is called twice, once along each of the edges meeting at the corner. If both boundary conditions at this point are Dirichlet and the prescribed solution values agree, then this value is used in an equation of the form (2). If the prescribed solution is discontinuous at the corner, then the average of the two values is used. If one boundary condition is Dirichlet and the other is mixed, then the value prescribed by the Dirichlet condition is used in an equation of the form given above. Finally, if both conditions are mixed or Neumann, then two 'fictitious' points are eliminated using two equations of the form (3).

It is possible that equations for which the solution is known at all points on the boundary, have coefficients which are not defined on the boundary. Since this routine calls the user-supplied subroutine PDEF at **all** points in the domain, including boundary points, arithmetic errors may occur in the user's routine PDEF which this routine cannot trap. If the user has an equation with Dirichlet boundary conditions (i.e., B = 0 at all points on the boundary), but with PDE coefficients which are singular on the boundary, then D03EDF could be called directly only using interior grid points with the user's own discretization.

After the equations have been set up as described above, they are checked for diagonal dominance. That is to say,

$$|A_{ij}^4| > \sum_{k\neq 4} |A_{ij}^k|, \quad i = 1, 2, \ldots, n_x;\ j = 1, 2, \ldots, n_y.$$

If this condition is not satisfied then the routine returns with IFAIL = 6. The multigrid routine D03EDF may still converge in this case, but if the coefficients of the first derivatives in the partial differential equation are large compared with the coefficients of the second derivative, the user should consider using upwind differences (SCHEME = 'U').

Since this routine is designed primarily for use with D03EDF, this document should be read in conjunction with the document for that routine.

4 References

[1] Wesseling P (1982) MGD1 – A robust and efficient multigrid method *Multigrid Methods. Lecture Notes in Mathematics* **960** Springer-Verlag 614–630

5 Parameters

1: XMIN — ***real*** *Input*

2: XMAX — ***real*** *Input*

On entry: the lower and upper x co-ordinates of the rectangular region respectively, x_A and x_B.

Constraint: XMIN < XMAX.

3: YMIN — ***real*** *Input*

4: YMAX — ***real*** *Input*

On entry: the lower and upper y co-ordinates of the rectangular region respectively, y_A and y_B.

Constraint: YMIN < YMAX.

5: PDEF — SUBROUTINE, supplied by the user. *External Procedure*

PDEF must evaluate the functions $\alpha(x,y)$, $\beta(x,y)$, $\gamma(x,y)$, $\delta(x,y)$, $\epsilon(x,y)$, $\phi(x,y)$ and $\psi(x,y)$ which define the equation at a general point (x,y).

Its specification is:

```
SUBROUTINE PDEF(X, Y, ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI, PSI)
real           X, Y, ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI, PSI
```

1: X — ***real*** *Input*

2: Y — ***real*** *Input*

On entry: the x and y co-ordinates of the point at which the coefficients of the partial differential equation are to be evaluated.

3: ALPHA — ***real*** *Output*

4: BETA — ***real*** *Output*

5: GAMMA — ***real*** *Output*

6: DELTA — ***real*** *Output*

7: EPSLON — ***real*** *Output*
8: PHI — ***real*** *Output*
9: PSI — ***real*** *Output*

On exit: ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI and PSI must be set to the values of $\alpha(x,y)$, $\beta(x,y)$, $\gamma(x,y)$, $\delta(x,y)$, $\epsilon(x,y)$, $\phi(x,y)$ and $\psi(x,y)$ respectively at the point specified by X and Y.

PDEF must be declared as EXTERNAL in the (sub)program from which D03EEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: BNDY — SUBROUTINE, supplied by the user. *External Procedure*

BNDY must evaluate the functions $a(x,y)$, $b(x,y)$, and $c(x,y)$ involved in the boundary conditions.

Its specification is:

```
SUBROUTINE BNDY(X, Y, A, B, C, IBND)
INTEGER         IBND
real            X, Y, A, B, C
```

1: X — ***real*** *Input*
2: Y — ***real*** *Input*

On entry: the x and y co-ordinates of the point at which the boundary conditions are to be evaluated.

3: A — ***real*** *Output*
4: B — ***real*** *Output*
5: C — ***real*** *Output*

On exit: A, B and C must be set to the values of the functions appearing in the boundary conditions.

6: IBND — INTEGER *Input*

On entry: specifies on which boundary the point (X,Y) lies. IBND = 0, 1, 2 or 3 according as the point lies on the bottom, right, top or left boundary.

BNDY must be declared as EXTERNAL in the (sub)program from which D03EEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: NGX — INTEGER *Input*
8: NGY — INTEGER *Input*

On entry: the number of interior grid points in the x- and y-directions respectively, n_x and n_y. If the seven-diagonal equations are to be solved by D03EDF, then NGX − 1 and NGY − 1 should preferably be divisible by as high a power of 2 as possible.

Constraint: NGX $\geq$ 3, NGY $\geq$ 3.

9: LDA — INTEGER *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which D03EEF is called.

Constraint: if only the seven-diagonal equations are required, then LDA $\geq$ NGX $\times$ NGY. If a call to this routine is to be followed by a call to D03EDF to solve the seven-diagonal linear equations, LDA $\geq (4 \times (\text{NGX}+1) \times (\text{NGY}+1))/3$.

Note. This routine only checks the former condition. D03EDF, if called, will check the latter condition.

10: A(LDA,7) — ***real*** array *Output*

On exit: A(i, j), for $i = 1, 2, \ldots,$NGX × NGY; j = 1,2,...,7, contains the seven-diagonal linear equations produced by the discretization described above. If LDA > NGX × NGY, the remaining elements are not referenced by the routine, but if LDA ≥ (4 × (NGX+1) × (NGY+1))/3 then the array A can be passed directly to D03EDF, where these elements are used as workspace.

11: RHS(LDA) — ***real*** array *Output*

On exit: the first NGX × NGY elements contain the right-hand sides of the seven-diagonal linear equations produced by the discretization described above. If LDA > NGX × NGY, the remaining elements are not referenced by the routine, but if LDA ≥ (4 × (NGY+1) × (NGY+1))/3 then the array RHS can be passed directly to D03EDF, where these elements are used as workspace.

12: SCHEME — CHARACTER*1 *Input*

On entry: the type of approximation to be used for the first derivatives which occur in the partial differential equation.

If SCHEME = 'C', then central differences are used.

If SCHEME = 'U', then upwind differences are used.

Constraint: SCHEME = 'C' or 'U'.

Note. Generally speaking, if at least one of the coefficients multiplying the first derivatives (DELTA or EPSLON as returned by PDEF) are large compared with the coefficients multiplying the second derivatives, then upwind differences may be more appropriate. Upwind differences are less accurate than central differences, but may result in more rapid convergence for strongly convective equations. The easiest test is to try both schemes.

13: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit**.

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 specified by the routine:

IFAIL = 1

On entry, XMIN ≥ XMAX,
or YMIN ≥ YMAX,
or NGX < 3,
or NGY < 3,
or LDA < NGX × NGY,
or SCHEME is not one of 'C' or 'U'.

IFAIL = 2

At some point on the boundary there is a derivative in the boundary conditions (B ≠ 0 on return from a BNDY) and there is a non-zero coefficient of the mixed derivative $\frac{\partial^2 U}{\partial x \partial y}$ (BETA ≠ 0 on return from PDEF).

IFAIL = 3

A null boundary has been specified, i.e., at some point both A and B are zero on return from a call to BNDY.

IFAIL = 4

The equation is not elliptic, i.e., 4 × ALPHA × GAMMA < BETA^2 after a call to PDEF. The discretization has been completed, but the convergence of D03EDF cannot be guaranteed.

IFAIL = 5

The boundary conditions are purely Neumann (only the derivative is specified) and there is, in general, no unique solution.

IFAIL = 6

The equations were not diagonally dominant. (See Section 3).

7 Accuracy

Not applicable.

8 Further Comments

If this routine is used as a pre-processor to the multigrid routine D03EDF it should be noted that the rate of convergence of that routine is strongly dependent upon the number of levels in the multigrid scheme, and thus the choice of NGX and NGY is very important.

9 Example

The program solves the elliptic partial differential equation

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + 50\left\{\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y}\right\} = f(x,y)$$

on the unit square $0 \leq x,\ y \leq 1$, with boundary conditions

$\frac{\partial U}{\partial n}$ given on $x = 0$ and $y = 0$,

U given on $x = 1$ and $y = 1$.

The function $f(x,y)$ and the exact form of the boundary conditions are derived from the exact solution $U(x,y) = \sin x \sin y$.

The equation is first solved using central differences. Since the coefficients of the first derivatives are large, the linear equations are not diagonally dominated, and convergence is slow. The equation is solved a second time with upwind differences, showing that convergence is more rapid, but the solution is less accurate.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     D03EEF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LEVELS, NGX, NGY, LDA
      PARAMETER        (LEVELS=3,NGX=2**LEVELS+1,NGY=NGX,LDA=4*(NGX+1)
```

```
     +                 *(NGY+1)/3)
*     .. Arrays in Common ..
      real              USER(6)
*     .. Local Scalars ..
      real              ACC, HX, HY, PI, RMSERR, XMAX, XMIN, YMAX, YMIN
      INTEGER           I, IFAIL, IOUT, J, MAXIT, NUMIT
*     .. Local Arrays ..
      real              A(LDA,7), RHS(LDA), U(LDA), UB(NGX*NGY), US(LDA),
     +                  X(NGX*NGY), Y(NGX*NGY)
*     .. External Functions ..
      real              FEXACT, X01AAF
      EXTERNAL          FEXACT, X01AAF
*     .. External Subroutines ..
      EXTERNAL          BNDY, D03EDF, D03EEF, PDEF
*     .. Intrinsic Functions ..
      INTRINSIC         real, SQRT
*     .. Common blocks ..
      COMMON            /BLOCK1/USER
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03EEF Example Program Results'
      WRITE (NOUT,*)
      PI = X01AAF(0.0e0)
*
*     USER(1) .. USER(6) contain the coefficients ALPHA, BETA, GAMMA,
*     DELTA, EPSLON and PHI appearing in the example partial
*     differential equation. They are stored in COMMON for use in PDEF.
*
      USER(1) = 1.0e0
      USER(2) = 0.0e0
      USER(3) = 1.0e0
      USER(4) = 50.0e0
      USER(5) = 50.0e0
      USER(6) = 0.0e0
*
      XMIN = 0.0e0
      XMAX = 1.0e0
      YMIN = 0.0e0
      YMAX = 1.0e0
      HX = (XMAX-XMIN)/real(NGX-1)
      HY = (YMAX-YMIN)/real(NGY-1)
      DO 40 I = 1, NGX
         DO 20 J = 1, NGY
            X(I+(J-1)*NGX) = XMIN + real(I-1)*HX
            Y(I+(J-1)*NGX) = YMIN + real(J-1)*HY
   20    CONTINUE
   40 CONTINUE
*
*     Discretize the equations
*
      IFAIL = -1
*
      CALL D03EEF(XMIN,XMAX,YMIN,YMAX,PDEF,BNDY,NGX,NGY,LDA,A,RHS,
     +            'Central',IFAIL)
*
*     Set the initial guess to zero
*
      DO 60 I = 1, NGX*NGY
         UB(I) = 0.0e0
```

```
   60 CONTINUE
*
*     Solve the equations
*
*     ** set IOUT.GE.2 to obtain intermediate output from D03EDF **
*
      IOUT = 0
      ACC = 1.0e-6
      MAXIT = 50
      IFAIL = -1
*
      CALL D03EDF(NGX,NGY,LDA,A,RHS,UB,MAXIT,ACC,US,U,IOUT,NUMIT,IFAIL)
*
*     Print out the solution
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Exact solution above computed solution'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) '  I/J', (I,I=1,NGX)
      RMSERR = 0.0e0
      DO 100 J = NGY, 1, -1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) J, (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX)),
     +     I=1,NGX)
         WRITE (NOUT,99999) J, (U(I+(J-1)*NGX),I=1,NGX)
         DO 80 I = 1, NGX
            RMSERR = RMSERR + (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX))
     +               -U(I+(J-1)*NGX))**2
   80    CONTINUE
  100 CONTINUE
      RMSERR = SQRT(RMSERR/real(NGX*NGY))
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Number of Iterations = ', NUMIT
      WRITE (NOUT,99996) 'RMS Error = ', RMSERR
*
*     Now discretize and solve the equations using upwind differences
*
      IFAIL = -1
*
      CALL D03EEF(XMIN,XMAX,YMIN,YMAX,PDEF,BNDY,NGX,NGY,LDA,A,RHS,
     +            'Upwind',IFAIL)
*
      IFAIL = -1
*
*     Set the initial guess to zero
*
      DO 120 I = 1, NGX*NGY
         UB(I) = 0.0e0
  120 CONTINUE
*
      CALL D03EDF(NGX,NGY,LDA,A,RHS,UB,MAXIT,ACC,US,U,IOUT,NUMIT,IFAIL)
*
*     Print the solution
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Exact solution above computed solution'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) '  I/J', (I,I=1,NGX)
```

```
      RMSERR = 0.0e0
      DO 160 J = NGY, 1, -1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) J, (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX)),
     +     I=1,NGX)
         WRITE (NOUT,99999) J, (U(I+(J-1)*NGX),I=1,NGX)
         DO 140 I = 1, NGX
            RMSERR = RMSERR + (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX))
     +               -U(I+(J-1)*NGX))**2
  140    CONTINUE
  160 CONTINUE
      RMSERR = SQRT(RMSERR/real(NGX*NGY))
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Number of Iterations = ', NUMIT
      WRITE (NOUT,99996) 'RMS Error = ', RMSERR
      STOP
*
99999 FORMAT (1X,I3,2X,10F7.3,:/(6X,10F7.3))
99998 FORMAT (1X,A,10I7,:/(6X,10I7))
99997 FORMAT (1X,A,I3)
99996 FORMAT (1X,A,1P,e10.2)
      END
*
      SUBROUTINE PDEF(X,Y,ALPHA,BETA,GAMMA,DELTA,EPSLON,PHI,PSI)
*     .. Scalar Arguments ..
      real              ALPHA, BETA, DELTA, EPSLON, GAMMA, PHI, PSI, X, Y
*     .. Arrays in Common ..
      real              USER(6)
*     .. Intrinsic Functions ..
      INTRINSIC         COS, SIN
*     .. Common blocks ..
      COMMON            /BLOCK1/USER
*     .. Executable Statements ..
      ALPHA = USER(1)
      BETA = USER(2)
      GAMMA = USER(3)
      DELTA = USER(4)
      EPSLON = USER(5)
      PHI = USER(6)
*
      PSI = (-ALPHA-GAMMA+PHI)*SIN(X)*SIN(Y) + BETA*COS(X)*COS(Y) +
     +      DELTA*COS(X)*SIN(Y) + EPSLON*SIN(X)*COS(Y)
*
      RETURN
      END
*
      SUBROUTINE BNDY(X,Y,A,B,C,IBND)
*     .. Parameters ..
      INTEGER           BOTTOM, RIGHT, TOP, LEFT
      PARAMETER         (BOTTOM=0,RIGHT=1,TOP=2,LEFT=3)
*     .. Scalar Arguments ..
      real              A, B, C, X, Y
      INTEGER           IBND
*     .. Intrinsic Functions ..
      INTRINSIC         SIN
*     .. Executable Statements ..
      IF (IBND.EQ.TOP .OR. IBND.EQ.RIGHT) THEN
*
```

```
*        Solution prescribed
*
         A = 1.0e0
         B = 0.0e0
         C = SIN(X)*SIN(Y)
      ELSE IF (IBND.EQ.BOTTOM) THEN
*
*        Derivative prescribed
*
         A = 0.0e0
         B = 1.0e0
         C = -SIN(X)
      ELSE IF (IBND.EQ.LEFT) THEN
*
*        Derivative prescribed
*
         A = 0.0e0
         B = 1.0e0
         C = -SIN(Y)
      END IF
*
      RETURN
      END
*
      real FUNCTION FEXACT(X,Y)
*     .. Scalar Arguments ..
      real                  X, Y
*     .. Intrinsic Functions ..
      INTRINSIC             SIN
*     .. Executable Statements ..
      FEXACT = SIN(X)*SIN(Y)
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D03EEF Example Program Results

** The linear equations were not diagonally dominated
** ABNORMAL EXIT from NAG Library routine D03EEF: IFAIL =     6
** NAG soft failure - control returned

Exact solution above computed solution

  I/J      1      2      3      4      5      6      7      8      9

  9     0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708
  9     0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708

  8     0.000  0.096  0.190  0.281  0.368  0.449  0.523  0.589  0.646
  8     0.000  0.095  0.190  0.281  0.368  0.449  0.523  0.589  0.646

  7     0.000  0.085  0.169  0.250  0.327  0.399  0.465  0.523  0.574
  7     0.000  0.084  0.168  0.249  0.326  0.398  0.464  0.523  0.574
```

```
  6    0.000  0.073  0.145  0.214  0.281  0.342  0.399  0.449  0.492
  6   -0.001  0.072  0.144  0.213  0.280  0.342  0.398  0.449  0.492

  5    0.000  0.060  0.119  0.176  0.230  0.281  0.327  0.368  0.403
  5   -0.001  0.059  0.118  0.174  0.229  0.280  0.326  0.368  0.403

  4    0.000  0.046  0.091  0.134  0.176  0.214  0.250  0.281  0.308
  4   -0.001  0.044  0.089  0.133  0.174  0.213  0.249  0.281  0.308

  3    0.000  0.031  0.061  0.091  0.119  0.145  0.169  0.190  0.208
  3   -0.001  0.029  0.060  0.089  0.118  0.144  0.168  0.190  0.208

  2    0.000  0.016  0.031  0.046  0.060  0.073  0.085  0.096  0.105
  2   -0.001  0.014  0.029  0.044  0.059  0.072  0.084  0.095  0.105

  1    0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  1   -0.001 -0.001 -0.001 -0.001 -0.001 -0.001  0.000  0.000  0.000

Number of Iterations =  10
RMS Error =   7.92E-04

Exact solution above computed solution

  I/J      1      2      3      4      5      6      7      8      9

  9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708
  9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708

  8    0.000  0.096  0.190  0.281  0.368  0.449  0.523  0.589  0.646
  8   -0.002  0.093  0.186  0.276  0.362  0.443  0.517  0.585  0.646

  7    0.000  0.085  0.169  0.250  0.327  0.399  0.465  0.523  0.574
  7   -0.005  0.078  0.160  0.239  0.316  0.388  0.455  0.517  0.574

  6    0.000  0.073  0.145  0.214  0.281  0.342  0.399  0.449  0.492
  6   -0.008  0.063  0.132  0.200  0.266  0.329  0.388  0.443  0.492

  5    0.000  0.060  0.119  0.176  0.230  0.281  0.327  0.368  0.403
  5   -0.011  0.047  0.103  0.159  0.214  0.266  0.316  0.362  0.403

  4    0.000  0.046  0.091  0.134  0.176  0.214  0.250  0.281  0.308
  4   -0.013  0.030  0.074  0.117  0.159  0.200  0.239  0.276  0.308

  3    0.000  0.031  0.061  0.091  0.119  0.145  0.169  0.190  0.208
  3   -0.015  0.014  0.044  0.074  0.103  0.132  0.160  0.186  0.208

  2    0.000  0.016  0.031  0.046  0.060  0.073  0.085  0.096  0.105
  2   -0.016 -0.001  0.014  0.030  0.047  0.063  0.078  0.093  0.105

  1    0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  1   -0.016 -0.016 -0.015 -0.013 -0.011 -0.008 -0.005 -0.002  0.000

Number of Iterations =   4
RMS Error =   1.05E-02
```