NAG Library Routine Document

D03EEF

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, LDA, A,    &
                   RHS, SCHEME, IFAIL)

INTEGER             NGX, NGY, LDA, IFAIL
REAL (KIND=nag_wp) 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 $\dfrac{\partial U}{\partial n}$ 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 coordinates 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.

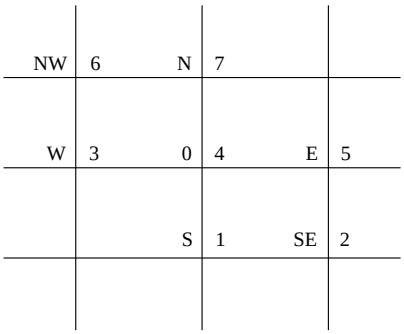

Figure 1

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.

The following approximations are used for the second derivatives:

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

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

Central Differences

$$\begin{aligned}
\frac{\partial U}{\partial x} &\simeq \tfrac{1}{2h_x}(u_{\mathrm{E}} - u_{\mathrm{W}}) \\
\frac{\partial U}{\partial y} &\simeq \tfrac{1}{2h_y}(u_{\mathrm{N}} - u_{\mathrm{S}})
\end{aligned}$$

Upwind Differences

$$\begin{aligned}
\frac{\partial U}{\partial x} &\simeq \tfrac{1}{h_x}(u_{\mathrm{O}} - u_{\mathrm{W}}) \quad \text{if} \quad \delta(x,y) > 0 \\
\frac{\partial U}{\partial x} &\simeq \tfrac{1}{h_x}(u_{\mathrm{E}} - u_{\mathrm{O}}) \quad \text{if} \quad \delta(x,y) < 0 \\
\frac{\partial U}{\partial y} &\simeq \tfrac{1}{h_y}(u_{\mathrm{N}} - u_{\mathrm{O}}) \quad \text{if} \quad \epsilon(x,y) > 0 \\
\frac{\partial U}{\partial y} &\simeq \tfrac{1}{h_y}(u_{\mathrm{O}} - u_{\mathrm{S}}) \quad \text{if} \quad \epsilon(x,y) < 0.
\end{aligned}$$

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:

$$\begin{aligned}
\frac{\partial U}{\partial x} &\simeq \tfrac{1}{2h_x}((k_x - 1)u_{\mathrm{W}} - 2k_x u_{\mathrm{O}} + (k_x + 1)u_{\mathrm{E}}) \\
\frac{\partial U}{\partial y} &\simeq \tfrac{1}{2h_y}\big((k_y - 1)u_{\mathrm{S}} - 2k_y u_{\mathrm{O}} + (k_y + 1)u_{\mathrm{N}}\big)
\end{aligned}$$

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 PDEF, and applying the approximations. This leads to a seven-diagonal system of linear equations of the form:

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

where the coefficients are given by

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

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:

$$\mu u_{ij} = \mu f_{ij}, \mu \neq 0. \tag{2}$$

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 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}:

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

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. 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 PDEF at **all** points in the domain, including boundary points, arithmetic errors may occur in PDEF which this routine cannot trap. If you have 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 at your discretization.

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

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

If this condition is not satisfied then the routine returns with IFAIL $= 6$. The multigrid routineD03EDF 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, you 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

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

5 Parameters

1: XMIN – REAL (KIND=nag_wp) *Input*
2: XMAX – REAL (KIND=nag_wp) *Input*

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

Constraint: XMIN $<$ XMAX.

3: YMIN – REAL (KIND=nag_wp) *Input*
4: YMAX – REAL (KIND=nag_wp) *Input*

On entry: the lower and upper y coordinates 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).

The specification of PDEF is:

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

REAL (KIND=nag_wp) X, Y, ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI,     &
                   PSI
```

1: X – REAL (KIND=nag_wp) *Input*
2: Y – REAL (KIND=nag_wp) *Input*

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

3: ALPHA – REAL (KIND=nag_wp) *Output*
4: BETA – REAL (KIND=nag_wp) *Output*
5: GAMMA – REAL (KIND=nag_wp) *Output*
6: DELTA – REAL (KIND=nag_wp) *Output*
7: EPSLON – REAL (KIND=nag_wp) *Output*
8: PHI – REAL (KIND=nag_wp) *Output*
9: PSI – REAL (KIND=nag_wp) *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 either be a module subprogram USEd by, or 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.

The specification of BNDY is:

```
SUBROUTINE BNDY (X, Y, A, B, C, IBND)
INTEGER            IBND
REAL (KIND=nag_wp) X, Y, A, B, C
```

1: X – REAL (KIND=nag_wp) *Input*
2: Y – REAL (KIND=nag_wp) *Input*

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

3: A – REAL (KIND=nag_wp) *Output*
4: B – REAL (KIND=nag_wp) *Output*
5: C – REAL (KIND=nag_wp) *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 either be a module subprogram USEd by, or 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.

Constraints:

NGX ≥ 3;
NGY ≥ 3.

9: LDA – INTEGER *Input*

On entry: the first dimension of the array A and the dimension of the array RHS 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 (KIND=nag_wp) array *Output*

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

11: RHS(LDA) – REAL (KIND=nag_wp) array *Output*

On exit: the first NGX $\times$ NGY elements contain the right-hand sides of the seven-diagonal linear equations produced by the discretization described above. If LDA $>$ NGX $\times$ NGY, the remaining elements are not referenced by the routine, but if LDA $\geq (4 \times (\text{NGY} + 1) \times (\text{NGY} + 1))/3$ 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.

SCHEME = 'C'
Central differences are used.

SCHEME = 'U'
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. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction 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, because for this routine the values of the output parameters may be useful even if IFAIL $\neq 0$ on exit, the recommended value is -1. **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).

Note: D03EEF may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, XMIN $\geq$ XMAX,
or YMIN $\geq$ YMAX,
or NGX < 3,
or NGY < 3,
or LDA $<$ NGX $\times$ 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 $\neq 0$ on return from BNDY) and there is a nonzero coefficient of the mixed derivative $\dfrac{\partial^2 U}{\partial x \partial y}$ (BETA $\neq 0$ 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., $4 \times$ ALPHA $\times$ 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.)

IFAIL $= -99$

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

See Section 3.8 in the Essential Introduction for further information.

IFAIL $= -399$

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

See Section 3.7 in the Essential Introduction for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.6 in the Essential Introduction for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

If this routine is used as a preprocessor 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.

10 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.

10.1 Program Text

```
!   D03EEF Example Program Text
!   Mark 25 Release. NAG Copyright 2014.

    Module d03eefe_mod

!     D03EEF 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                                :: bndy, fexact, pdef
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter         :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Integer, Parameter, Public            :: nin = 5, nout = 6
    Contains
      Subroutine pdef(x,y,alpha,beta,gamma,delta,epslon,phi,psi)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: alpha, beta, delta, epslon,    &
                                                gamma, phi, psi
        Real (Kind=nag_wp), Intent (In)      :: x, y
!       .. Intrinsic Procedures ..
```

```
        Intrinsic                           :: cos, sin
!       .. Executable Statements ..
        alpha = one
        beta = zero
        gamma = one
        delta = 50.0_nag_wp
        epslon = 50.0_nag_wp
        phi = zero

        psi = sin(x)*((-alpha-gamma+phi)*sin(y)+epslon*cos(y)) + &
          cos(x)*(delta*sin(y)+beta*cos(y))

        Return
      End Subroutine pdef
      Subroutine bndy(x,y,a,b,c,ibnd)

!       .. Parameters ..
        Integer, Parameter                  :: bottom = 0, left = 3,          &
                                               right = 1, top = 2
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)    :: a, b, c
        Real (Kind=nag_wp), Intent (In)     :: x, y
        Integer, Intent (In)                :: ibnd
!       .. Intrinsic Procedures ..
        Intrinsic                           :: sin
!       .. Executable Statements ..
        If (ibnd==top .Or. ibnd==right) Then

!         Solution prescribed

          a = one
          b = zero
          c = sin(x)*sin(y)
        Else If (ibnd==bottom) Then

!         Derivative prescribed

          a = zero
          b = one
          c = -sin(x)
        Else If (ibnd==left) Then

!         Derivative prescribed

          a = zero
          b = one
          c = -sin(y)
        End If

        Return
      End Subroutine bndy
      Function fexact(x,y)

!       .. Function Return Value ..
        Real (Kind=nag_wp)                  :: fexact
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)     :: x, y
!       .. Intrinsic Procedures ..
        Intrinsic                           :: sin
!       .. Executable Statements ..
        fexact = sin(x)*sin(y)
        Return
      End Function fexact
    End Module d03eefe_mod

    Program d03eefe

!     D03EEF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d03edf, d03eef, nag_wp
```

```
      Use d03eefe_mod, Only: bndy, fexact, nin, nout, pdef, zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: acc, hx, hy, rmserr, xmax, xmin, &
                                             xx, ymax, ymin, yy
      Integer                             :: i, icase, ifail, iout, ix, j,    &
                                             lda, levels, maxit, ngx, ngxy,   &
                                             ngy, numit
      Character (7)                       :: scheme
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: a(:,:), rhs(:), u(:), ub(:),     &
                                             us(:), x(:), y(:)
!     .. Intrinsic Procedures ..
      Intrinsic                           :: real, sqrt
!     .. Executable Statements ..
      Write (nout,*) 'D03EEF Example Program Results'
      Write (nout,*)
      Flush (nout)

!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) levels
      ngx = 2**levels + 1
      ngy = ngx
      lda = 4*(ngx+1)*(ngy+1)/3
      ngxy = ngx*ngy
      Allocate (a(lda,7),rhs(lda),u(lda),ub(ngxy),us(lda),x(ngxy),y(ngxy))
      Read (nin,*) xmin, xmax
      Read (nin,*) ymin, ymax
      hx = (xmax-xmin)/real(ngx-1,kind=nag_wp)
      Do i = 1, ngx
        xx = xmin + real(i-1,kind=nag_wp)*hx
        x(i:ngxy:ngx) = xx
      End Do
      hy = (ymax-ymin)/real(ngy-1,kind=nag_wp)
      Do j = 1, ngy
        yy = ymin + real(j-1,kind=nag_wp)*hy
        y((j-1)*ngx+1:j*ngx) = yy
      End Do
!     ** set iout > 2 to obtain intermediate output from D03EDF **
      iout = 0
      Read (nin,*) acc
      Read (nin,*) maxit


cases: Do icase = 1, 2

        Select Case (icase)
        Case (1)
!         Central differences
          scheme = 'Central'
        Case (2)
!         Upwind differences
          scheme = 'Upwind'
        End Select
!       Discretize the equations
!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = -1
        Call d03eef(xmin,xmax,ymin,ymax,pdef,bndy,ngx,ngy,lda,a,rhs,scheme, &
          ifail)

        If (ifail<0) Then
          Write (nout,99995) ifail
          Exit cases
        End If

!       Set the initial guess to zero
        ub(1:ngxy) = zero
```

```
!       Solve the equations
        ifail = 0
        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 = zero
        Do j = ngy, 1, -1
          ix = (j-1)*ngx
          Write (nout,*)
          Write (nout,99999) j, (fexact(x(ix+i),y(ix+i)),i=1,ngx)
          Write (nout,99999) j, u(ix+1:ix+ngx)
          Do i = 1, ngx
            rmserr = rmserr + (fexact(x(ix+i),y(ix+i))-u(ix+i))**2
          End Do
        End Do
        rmserr = sqrt(rmserr/real(ngxy,kind=nag_wp))
        Write (nout,*)
        Write (nout,99997) 'Number of Iterations = ', numit
        Write (nout,99996) 'RMS Error = ', rmserr
      End Do cases

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)
99995 Format (1X,' ** D03EEF returned with IFAIL = ',I5)
    End Program d03eefe
```

10.2 Program Data

```
D03EEF Example Program Data
  3                          : levels
  0.0 1.0                    : xmin, xmax
  0.0 1.0                    : ymin, ymax
  1.0E-6                     : acc
  50                         : maxit
```

10.3 Program Results

```
 D03EEF Example Program Results

 ** The linear equations were not diagonally dominant.
 ** 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
```

Example Program
Solution of Elliptic PDE using Central Differences