NAG Library Routine Document

D03ECF

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

D03ECF uses the Strongly Implicit Procedure to calculate the solution to a system of simultaneous algebraic equations of seven-point molecule form on a three-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example, can be used if it is equivalent to a rectangular box.)

2 Specification

```
SUBROUTINE D03ECF (N1, N2, N3, LDA, SDA, A, B, C, D, E, F, G, Q, T,        &
                   APARAM, ITMAX, ITCOUN, ITUSED, NDIR, IXN, IYN, IZN,     &
                   CONRES, CONCHN, RESIDS, CHNGS, WRKSP1, WRKSP2,          &
                   WRKSP3, WRKSP4, IFAIL)

INTEGER            N1, N2, N3, LDA, SDA, ITMAX, ITCOUN, ITUSED, NDIR,      &
                   IXN, IYN, IZN, IFAIL
REAL (KIND=nag_wp) A(LDA,SDA,N3), B(LDA,SDA,N3), C(LDA,SDA,N3),            &
                   D(LDA,SDA,N3), E(LDA,SDA,N3), F(LDA,SDA,N3),            &
                   G(LDA,SDA,N3), Q(LDA,SDA,N3), T(LDA,SDA,N3),            &
                   APARAM, CONRES, CONCHN, RESIDS(ITMAX),                  &
                   CHNGS(ITMAX), WRKSP1(LDA,SDA,N3),                       &
                   WRKSP2(LDA,SDA,N3), WRKSP3(LDA,SDA,N3),                 &
                   WRKSP4(LDA,SDA,N3)
```

3 Description

Given a set of simultaneous equations

$$Mt = q \tag{1}$$

(which could be nonlinear) derived, for example, from a finite difference representation of a three-dimensional elliptic partial differential equation and its boundary conditions, the routine determines the values of the dependent variable t. M is a square $(n_1 \times n_2 \times n_3)$ by $(n_1 \times n_2 \times n_3)$ matrix and q is a known vector of length $(n_1 \times n_2 \times n_3)$.

The equations must be of seven-diagonal form:

$$a_{ijk}t_{ij,k-1} + b_{ijk}t_{i,j-1,k} + c_{ijk}t_{i-1,jk} + d_{ijk}t_{ijk} + e_{ijk}t_{i+1,jk} + f_{ijk}t_{i,j+1,k} + g_{ijk}t_{ij,k+1} = q_{ijk}$$

for $i = 1, 2, \ldots, n_1$, $j = 1, 2, \ldots, n_2$ and $k = 1, 2, \ldots, n_3$, provided that $d_{ijk} \neq 0.0$.

Indeed, if $d_{ijk} = 0.0$, then the equation is assumed to be:

$$t_{ijk} = q_{ijk}.$$

The system is solved iteratively from a starting approximation $t^{(1)}$ by the formulae:

$$\begin{aligned} r^{(n)} &= q - Mt^{(n)} \\ Ms^{(n)} &= r^{(n)} \\ t^{(n+1)} &= t^{(n)} + s^{(n)}. \end{aligned}$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the update change vector.

The calling program supplies an initial approximation for the values of the dependent variable in the array T, the coefficients of the seven-point molecule system of equations in the arrays A, B, C, D, E, F and G, and the source terms in the array Q. The routine derives the residual of the latest approximate

solution, and then uses the approximate LU factorization of the Strongly Implicit Procedure with the necessary acceleration argument adjustment by calling D03UBF at each iteration. D03ECF combines the newly derived change with the old approximation to obtain the new approximate solution for t. The new solution is checked for convergence against the user-supplied convergence criteria, and if these have not been satisfied, the iterative cycle is repeated. Convergence is based on both the maximum absolute normalized residuals (calculated with reference to the previous approximate solution as these are calculated at the commencement of each iteration) and on the maximum absolute change made to the values of t.

Problems in topologically non-rectangular-box-shaped regions can be solved using the routine by surrounding the region by a circumscribing topologically rectangular box. The equations for the nodal values external to the region of interest are set to zero (i.e., $d_{ijk} = t_{ijk} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, one can use an array of zeros as the initial approximation.

The routine can be used to solve linear elliptic equations in which case the arrays A, B, C, D, E, F, G and Q remain constant and for which a single call provides the required solution. It can also be used to solve nonlinear elliptic equations, in which case some or all of these arrays may require updating during the progress of the iterations as more accurate solutions are derived. The routine will then have to be called repeatedly in an outer iterative cycle. Dependent on the nonlinearity, some under-relaxation of the coefficients and/or source terms may be needed during their recalculation using the new estimates of the solution.

The routine can also be used to solve each step of a time-dependent parabolic equation in three space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank–Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive-definiteness, of the matrix M formed from the arrays A, B, C, D, E, F and G is necessary to ensure convergence.

For problems in which the solution is not unique in the sense that an arbitrary constant can be added to the solution (for example Poisson's equation with all Neumann boundary conditions), an argument is incorporated so that the solution can be rescaled. A specified nodal value is subtracted from the whole solution t after the completion of every iteration. This keeps rounding errors to a minimum for those cases when convergence is slow. For such problems there is generally an associated compatibility condition. For the example mentioned this compatibility condition equates the total net source within the region (i.e., the source integrated over the region) with the total net outflow across the boundaries defined by the Neumann conditions (i.e., the normal derivative integrated along the whole boundary). It is very important that the algebraic equations derived to model such a problem implement accurately the compatibility condition. If they do not, a net source or sink is very likely to be represented by the set of algebraic equations and no steady-state solution of the equations exists.

4 References

Jacobs D A H (1972) The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations *Note RD/L/N66/72* Central Electricity Research Laboratory

Stone H L (1968) Iterative solution of implicit approximations of multi-dimensional partial differential equations *SIAM J. Numer. Anal.* **5** 530–558

Weinstein H G, Stone H L and Kwan T V (1969) Iterative procedure for solution of systems of parabolic and elliptic equations in three dimensions *Industrial and Engineering Chemistry Fundamentals* **8** 281–287

5 Arguments

1: N1 – INTEGER *Input*

On entry: the number of nodes in the first coordinate direction, n_1.

Constraint: N1 > 1.

2: N2 – INTEGER *Input*

On entry: the number of nodes in the second coordinate direction, n_2.

Constraint: N2 > 1.

3: N3 – INTEGER *Input*

On entry: the number of nodes in the third coordinate direction, n_3.

Constraint: N3 > 1.

4: LDA – INTEGER *Input*

On entry: the first dimension of the arrays A, B, C, D, E, F, G, Q, T, WRKSP1, WRKSP2, WRKSP3 and WRKSP4 as declared in the (sub)program from which D03ECF is called.

Constraint: LDA $\geq$ N1.

5: SDA – INTEGER *Input*

On entry: the second dimension of the arrays A, B, C, D, E, F, G, Q, T, WRKSP1, WRKSP2, WRKSP3 and WRKSP4 as declared in the (sub)program from which D03ECF is called.

Constraint: SDA $\geq$ N2.

6: A(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: A(i,j,k) must contain the coefficient of $t_{ij,k-1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of A, for $k = 1$, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

7: B(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: B(i,j,k) must contain the coefficient of $t_{i,j-1,k}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of B, for $j = 1$, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

8: C(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: C(i,j,k) must contain the coefficient of $t_{i-1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of C, for $i = 1$, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

9: D(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: D(i,j,k) must contain the coefficient of t_{ijk} (the 'central' term) in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of D are checked to ensure that they are nonzero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $t_{ijk} = q_{ijk}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest. Setting D$(i,j,k) = 0.0$ at appropriate points, and the corresponding value of Q(i,j,k) to the appropriate value, namely the prescribed value of T(i,j,k) in the Dirichlet case, or to zero at an external point.

10: E(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: E(i,j,k) must contain the coefficient of $t_{i+1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of E, for $i =$ N1, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

11: F(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: F(i,j,k) must contain the coefficient of $t_{i,j+1,k}$ in the (i,j,k)th equation of the system (1), , for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of F, for $j =$ N2, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

12: G(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: G(i,j,k) must contain the coefficient of $t_{ij,k+1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of G, for $k =$ N3, must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

13: Q(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input*

On entry: Q(i,j,k) must contain q_{ijk}, for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3, i. e., the source-term values at the nodal points of the system (1).

14: T(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Input/Output*

On entry: T(i,j,k) must contain the element t_{ijk} of an approximate solution to the equations, for $i = 1,2,\ldots,$N1, $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3.

If no better approximation is known, an array of zeros can be used.

On exit: the solution derived by the routine.

15: APARAM – REAL (KIND=nag_wp) *Input*

On entry: the iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: $0.0 <$ APARAM $\leq \left((\text{N1}-1)^2 + (\text{N2}-1)^2 + (\text{N3}-1)^2\right)/3.0$.

16: ITMAX – INTEGER *Input*

On entry: the maximum number of iterations to be used by the routine in seeking the solution. A reasonable value might be 20 for a problem with 3000 nodes and convergence criteria of about 10^{-3} of the original residual and change.

17: ITCOUN – INTEGER *Input/Output*

On entry: on the first call of D03ECF, ITCOUN must be set to 0. On subsequent entries, its value must be unchanged from the previous call.

On exit: its value is increased by the number of iterations used on this call (namely ITUSED). It therefore stores the accumulated number of iterations actually used.

For subsequent calls for the same problem, i.e., with the same N1, N2 and N3 but possibly different coefficients and/or source terms, as occur with nonlinear systems or with time-dependent systems, ITCOUN should not be reset, i.e., it must contain the accumulated number of iterations. In this way a suitable cycling of the sequence of iteration arguments is obtained in the calls to D03UBF.

18: ITUSED – INTEGER *Output*

On exit: the number of iterations actually used on that call.

19: NDIR – INTEGER *Input*

On entry: indicates whether or not the system of equations has a unique solution. For systems which have a unique solution, NDIR must be set to any nonzero value. For systems derived from

problems to which an arbitrary constant can be added to the solution, for example Poisson's equation with all Neumann boundary conditions, NDIR should be set to 0 and the values of the next three arguments must be specified. For such problems the routine subtracts the value of the function derived at the node (IXN, IYN, IZN) from the whole solution after each iteration to reduce the possibility of large rounding errors. You must also ensure for such problems that the appropriate compatibility condition on the source terms Q is satisfied. See the comments at the end of Section 3.

20: IXN – INTEGER *Input*

On entry: is ignored unless NDIR is equal to zero, in which case it must specify the first index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

21: IYN – INTEGER *Input*

On entry: is ignored unless NDIR is equal to zero, in which case it must specify the second index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

22: IZN – INTEGER *Input*

On entry: is ignored unless NDIR is equal to zero, in which case it must specify the third index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

23: CONRES – REAL (KIND=nag_wp) *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the normalized residual vector components. The latter is defined as the residual of the algebraic equation divided by the central coefficient when the latter is not equal to 0.0, and defined as the residual when the central coefficient is zero.

CONRES should not be less than a reasonable multiple of the ***machine precision***.

24: CONCHN – REAL (KIND=nag_wp) *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the change made at each iteration to the elements of the array T, namely the dependent variable. CONCHN should not be less than a reasonable multiple of the machine accuracy multiplied by the maximum value of T attained.

Convergence is achieved when both the convergence criteria are satisfied. You can therefore set convergence on either the residual or on the change, or (as is recommended) on a requirement that both are below prescribed limits.

25: RESIDS(ITMAX) – REAL (KIND=nag_wp) array *Output*

On exit: the maximum absolute value of the residuals calculated at the ith iteration, for $i = 1, 2, \ldots,$ ITUSED. If the residual of the solution is sought you must calculate this in the subroutine from which D03ECF is called. The sequence of values RESIDS indicates the rate of convergence.

26: CHNGS(ITMAX) – REAL (KIND=nag_wp) array *Output*

On exit: the maximum absolute value of the changes made to the components of the dependent variable T at the ith iteration, for $i = 1, 2, \ldots,$ ITUSED. The sequence of values CHNGS indicates the rate of convergence.

27: WRKSP1(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Workspace*
28: WRKSP2(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Workspace*
29: WRKSP3(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Workspace*
30: WRKSP4(LDA, SDA, N3) – REAL (KIND=nag_wp) array *Workspace*

31: 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, because for this routine the values of the output arguments 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: D03ECF 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, N1 < 2,
or N2 < 2,
or N3 < 2.

IFAIL $= 2$

On entry, LDA $<$ N1,
or SDA $<$ N2.

IFAIL $= 3$

On entry, APARAM ≤ 0.0.

IFAIL $= 4$

On entry, APARAM $> \left((\mathrm{N1}-1)^2+(\mathrm{N2}-1)^2+(\mathrm{N3}-1)^2\right)/3.0$.

IFAIL $= 5$

Convergence was not achieved after ITMAX iterations.

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

The improvement in accuracy for each iteration depends on the size of the system and on the condition of the update matrix characterised by the seven-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration arguments. The convergence may become slow with very large problems. The final accuracy obtained may be judged approximately from the rate of convergence determined from the sequence of values returned in the arrays RESIDS and CHNGS and the magnitude of the maximum absolute value of the change vector on the last iteration stored in CHNGS(ITUSED).

8 Parallelism and Performance

D03ECF is not threaded in any implementation.

9 Further Comments

The time taken per iteration is approximately proportional to N1 $\times$ N2 $\times$ N3.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case is often associated with a near ill-conditioned matrix.

10 Example

This example solves Laplace's equation in a rectangular box with a non-uniform grid spacing in the x, y, and z coordinate directions and with Dirichlet boundary conditions specifying the function on the surfaces of the box equal to

$$e^{(1.0+x)/y(n_2)} \times \cos\left(\sqrt{2}y/y(n_2)\right) \times e^{(-1.0-z)/y(n_2)}.$$

Note that this is the same problem as that solved in the example for D03UBF. The differences in the maximum residuals obtained at each iteration between the two test runs are explained by the fact that in D03ECF the residual at each node is normalized by dividing by the central coefficient, whereas this normalization has not been used in the example program for D03UBF.

10.1 Program Text

```
    Program d03ecfe

!     D03ECF Example Program Text

!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: d03ecf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter    :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter    :: zero = 0.0_nag_wp
      Integer, Parameter               :: nin = 5, nout = 6
!     .. Local Scalars ..
      Real (Kind=nag_wp)               :: aparam, conchn, conres, root2, x1,   &
                                          x2, y1, y2, yy, z1, z2
      Integer                          :: i, ifail, itcoun, itmax, itused,     &
```

```
                                       ixn, iyn, izn, j, k, lda, n1, n2,    &
                                       n3, ndir, sda
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: a(:,:,:), b(:,:,:), c(:,:,:),        &
                                       chngs(:), d(:,:,:), e(:,:,:),        &
                                       f(:,:,:), g(:,:,:), q(:,:,:),        &
                                       resids(:), t(:,:,:), wrksp1(:,:,:),  &
                                       wrksp2(:,:,:), wrksp3(:,:,:),        &
                                       wrksp4(:,:,:), x(:), y(:), z(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: cos, exp, sqrt
!     .. Executable Statements ..
      Write (nout,*) 'D03ECF Example Program Results'
      Write (nout,*)
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) n1, n2, n3, itmax
      lda = n1
      sda = n2
      Allocate (a(lda,sda,n3),b(lda,sda,n3),c(lda,sda,n3),chngs(itmax),        &
        d(lda,sda,n3),e(lda,sda,n3),f(lda,sda,n3),g(lda,sda,n3),q(lda,sda,n3), &
        resids(itmax),t(lda,sda,n3),wrksp1(lda,sda,n3),wrksp2(lda,sda,n3),     &
        wrksp3(lda,sda,n3),wrksp4(lda,sda,n3),x(n1),y(n2),z(n3))

      Read (nin,*) x(1:n1)
      Read (nin,*) y(1:n2)
      Read (nin,*) z(1:n3)
      Read (nin,*) conres, conchn
      Read (nin,*) ndir
      root2 = sqrt(two)
      aparam = one
      itcoun = 0
!     Set up difference equation coefficients, source terms and
!     initial approximation.
      a(1:n1,1:n2,1:n3) = zero
      b(1:n1,1:n2,1:n3) = zero
      c(1:n1,1:n2,1:n3) = zero
      d(1:n1,1:n2,1:n3) = zero
      e(1:n1,1:n2,1:n3) = zero
      f(1:n1,1:n2,1:n3) = zero
      g(1:n1,1:n2,1:n3) = zero
      q(1:n1,1:n2,1:n3) = zero
      t(1:n1,1:n2,1:n3) = zero
!     Non-zero Specification for internal nodes
      Do k = 2, n3 - 1
        Do j = 2, n2 - 1
          Do i = 2, n1 - 1
            a(i,j,k) = two/((z(k)-z(k-1))*(z(k+1)-z(k-1)))
            g(i,j,k) = two/((z(k+1)-z(k))*(z(k+1)-z(k-1)))
            b(i,j,k) = two/((y(j)-y(j-1))*(y(j+1)-y(j-1)))
            f(i,j,k) = two/((y(j+1)-y(j))*(y(j+1)-y(j-1)))
            c(i,j,k) = two/((x(i)-x(i-1))*(x(i+1)-x(i-1)))
            e(i,j,k) = two/((x(i+1)-x(i))*(x(i+1)-x(i-1)))
            d(i,j,k) = -a(i,j,k) - b(i,j,k) - c(i,j,k) - e(i,j,k) - f(i,j,k) - &
              g(i,j,k)
          End Do
        End Do
      End Do
!     Non-zero specification for boundary nodes
      yy = one/y(n2)
      x1 = (x(1)+one)*yy
      x2 = (x(n1)+one)*yy
      Do j = 1, n2
        y1 = root2*y(j)*yy
        q(1,j,1:n3) = exp(x1)*cos(y1)*exp((-z(1:n3)-one)*yy)
        q(n1,j,1:n3) = exp(x2)*cos(y1)*exp((-z(1:n3)-one)*yy)
      End Do
      y1 = root2*y(1)*yy
      y2 = root2*y(n2)*yy
      Do i = 1, n1
        x1 = (x(i)+one)*yy
```

```
        q(i,1,1:n3) = exp(x1)*cos(y1)*exp((-z(1:n3)-one)*yy)
        q(i,n2,1:n3) = exp(x1)*cos(y2)*exp((-z(1:n3)-one)*yy)
      End Do
      z1 = (-z(1)-one)*yy
      z2 = (-z(n3)-one)*yy
      Do i = 1, n1
        x1 = (x(i)+one)*yy
        q(i,1:n2,1) = exp(x1)*cos(root2*y(1:n2)*yy)*exp(z1)
        q(i,1:n2,n3) = exp(x1)*cos(root2*y(1:n2)*yy)*exp(z2)
      End Do

!     ifail: behaviour on error exit
!            =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0
      Call d03ecf(n1,n2,n3,lda,sda,a,b,c,d,e,f,g,q,t,aparam,itmax,itcoun,      &
        itused,ndir,ixn,iyn,izn,conres,conchn,resids,chngs,wrksp1,wrksp2,      &
        wrksp3,wrksp4,ifail)

      Write (nout,*) 'Iteration      Maximum        Maximum'
      Write (nout,*) ' number        residual        change'
      If (itused/=0) Then
        Write (nout,99999)(i,resids(i),chngs(i),i=1,itused)
      End If
      Write (nout,*)
      Write (nout,*) 'Table of calculated function values'
      Write (nout,*)
      Write (nout,99998)
      Do k = 1, n3
        Do j = 1, n2
          Write (nout,99997) k, j, (i,t(i,j,k),i=1,n1)
        End Do
      End Do

99999 Format (2X,I3,9X,E11.4,4X,E11.4)
99998 Format (1X,'K  J  ',4(' (I      T   )'))
99997 Format (1X,I1,2X,I1,1X,4(1X,I3,2X,F8.3))
    End Program d03ecfe
```

10.2 Program Data

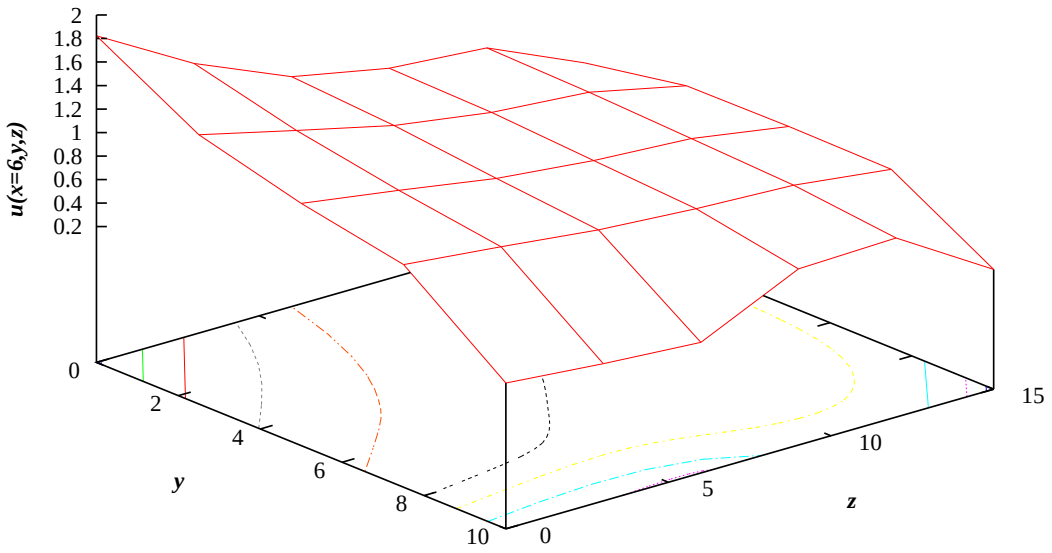
```
D03ECF Example Program Data
  4 5 6 18                    : n1, n2, n3, itmax
  0.0 1.0 3.0 6.0             : x
  0.0 1.0 3.0 6.0 10.0        : y
  0.0 1.0 3.0 6.0 10.0, 15.0  : z
  0.1E-5 0.1E-5               : conres, conchn
  1                           : ndir
```

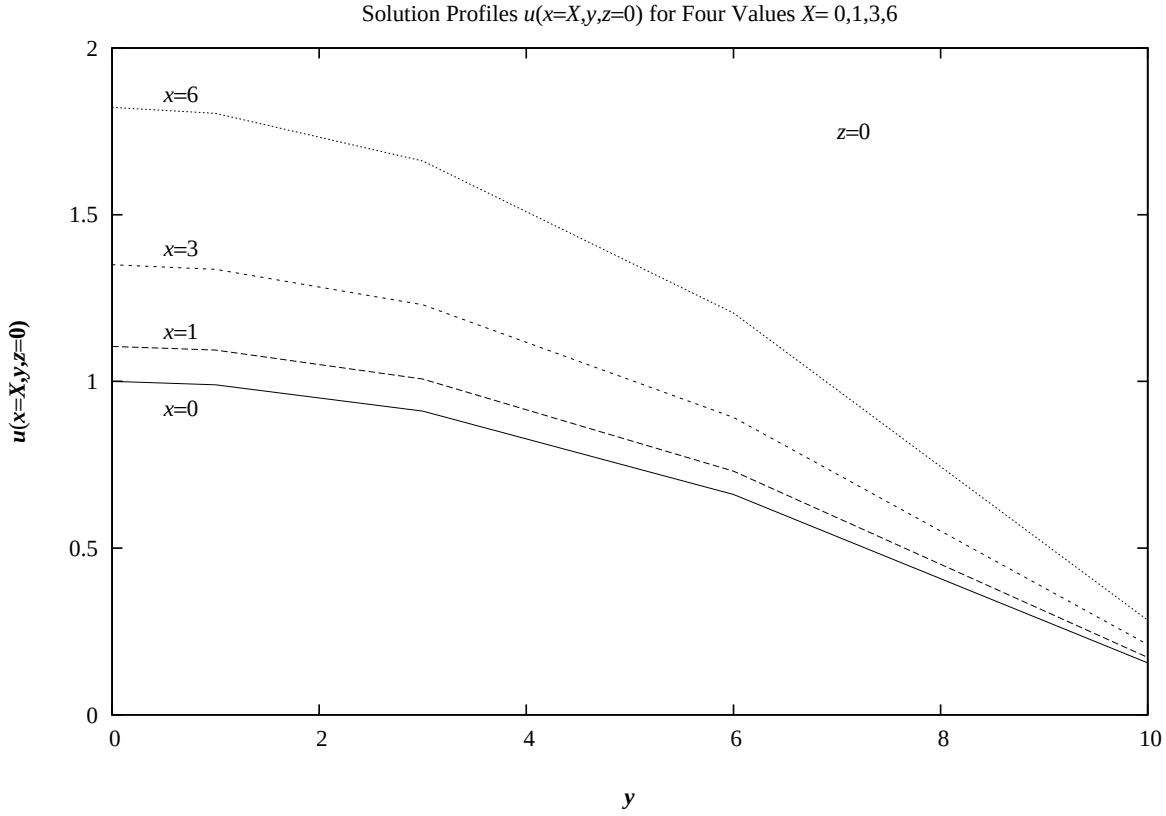
10.3 Program Results

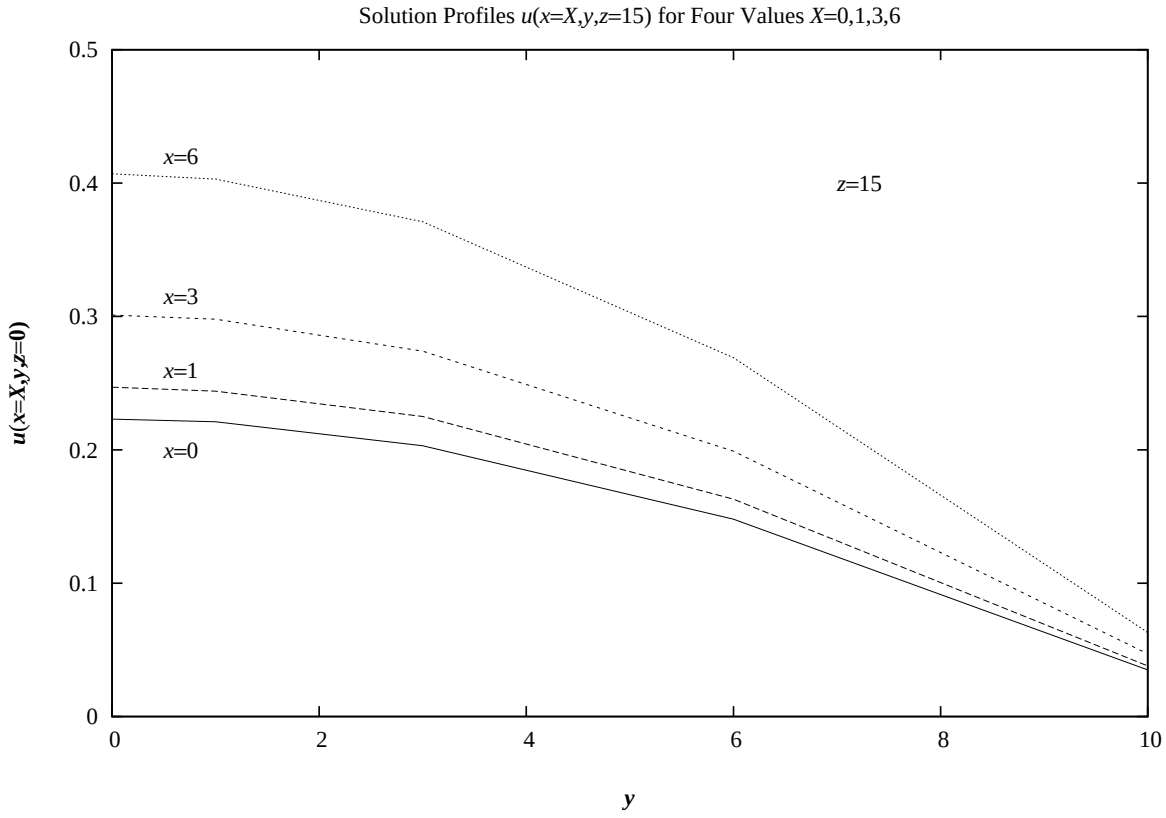
```
 D03ECF Example Program Results

 Iteration      Maximum        Maximum
  number        residual        change
    1         0.1822E+01     0.1822E+01
    2         0.9025E-02     0.1970E-01
    3         0.1358E-02     0.1496E-02
    4         0.4013E-04     0.3848E-04
    5         0.5321E-05     0.5481E-05
    6         0.2695E-06     0.2333E-06

 Table of calculated function values

 K  J   (I      T   ) (I      T   ) (I      T   ) (I      T   )
 1  1    1     1.000   2     1.105   3     1.350   4     1.822
 1  2    1     0.990   2     1.094   3     1.336   4     1.804
 1  3    1     0.911   2     1.007   3     1.230   4     1.661
 1  4    1     0.661   2     0.731   3     0.892   4     1.205
 1  5    1     0.156   2     0.172   3     0.211   4     0.284
 2  1    1     0.905   2     1.000   3     1.221   4     1.649
```

```
2   2     1      0.896    2      0.990    3      1.210    4      1.632
2   3     1      0.825    2      0.912    3      1.114    4      1.503
2   4     1      0.598    2      0.662    3      0.809    4      1.090
2   5     1      0.141    2      0.156    3      0.190    4      0.257
3   1     1      0.741    2      0.819    3      1.000    4      1.350
3   2     1      0.733    2      0.811    3      0.991    4      1.336
3   3     1      0.675    2      0.747    3      0.913    4      1.230
3   4     1      0.490    2      0.543    3      0.664    4      0.892
3   5     1      0.116    2      0.128    3      0.156    4      0.211
4   1     1      0.549    2      0.607    3      0.741    4      1.000
4   2     1      0.543    2      0.601    3      0.734    4      0.990
4   3     1      0.500    2      0.554    3      0.677    4      0.911
4   4     1      0.363    2      0.402    3      0.492    4      0.661
4   5     1      0.086    2      0.095    3      0.116    4      0.156
5   1     1      0.368    2      0.407    3      0.497    4      0.670
5   2     1      0.364    2      0.403    3      0.492    4      0.664
5   3     1      0.335    2      0.371    3      0.454    4      0.611
5   4     1      0.243    2      0.270    3      0.330    4      0.443
5   5     1      0.057    2      0.063    3      0.077    4      0.105
6   1     1      0.223    2      0.247    3      0.301    4      0.407
6   2     1      0.221    2      0.244    3      0.298    4      0.403
6   3     1      0.203    2      0.225    3      0.274    4      0.371
6   4     1      0.148    2      0.163    3      0.199    4      0.269
6   5     1      0.035    2      0.038    3      0.047    4      0.063
```

Example Program
Solution in Y-Z Planes for x=6




Solution Profiles u(x=X,y,z=0) for Four Values X= 0,1,3,6
x=6
x=3
x=1
x=0
z=0
u(x=X,y,z=0)
y
0
0.5
1
1.5
2
0
2
4
6
8
10


Solution Profiles u(x=X,y,z=15) for Four Values X=0,1,3,6
x=6
x=3
x=1
x=0
z=15
u(x=X,y,z=0)
y
0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10