NAG Library Routine Document

D06CCF

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

D06CCF renumbers the vertices of a given mesh using a Gibbs method, in order the reduce the bandwidth of Finite Element matrices associated with that mesh.

2 Specification

```
SUBROUTINE D06CCF (NV, NELT, NEDGE, NNZMAX, NNZ, COOR, EDGE, CONN, IROW,    &
                   ICOL, ITRACE, IWORK, LIWORK, RWORK, LRWORK, IFAIL)

INTEGER            NV, NELT, NEDGE, NNZMAX, NNZ, EDGE(3,NEDGE),             &
                   CONN(3,NELT), IROW(NNZMAX), ICOL(NNZMAX), ITRACE,        &
                   IWORK(LIWORK), LIWORK, LRWORK, IFAIL
REAL (KIND=nag_wp) COOR(2,NV), RWORK(LRWORK)
```

3 Description

D06CCF uses a Gibbs method to renumber the vertices of a given mesh in order to reduce the bandwidth of the associated finite element matrix A. This matrix has elements a_{ij} such that:

$$a_{ij} \neq 0 \Rightarrow i \text{ and } j \text{ are vertices belonging to the same triangle.}$$

This routine reduces the bandwidth m, which is the smallest integer such that $a_{ij} \neq 0$ whenever $|i-j| > m$ (see Gibbs *et al.* (1976) for details about that method). D06CCF also returns the sparsity structure of the matrix associated with the renumbered mesh.

This routine is derived from material in the MODULEF package from INRIA (Institut National de Recherche en Informatique et Automatique).

4 References

Gibbs N E, Poole W G Jr and Stockmeyer P K (1976) An algorithm for reducing the bandwidth and profile of a sparse matrix *SIAM J. Numer. Anal.* **13** 236–250

5 Arguments

1: NV – INTEGER *Input*

On entry: the total number of vertices in the input mesh.

Constraint: NV ≥ 3.

2: NELT – INTEGER *Input*

On entry: the number of triangles in the input mesh.

Constraint: NELT $\leq 2 \times$ NV $- 1$.

3: NEDGE – INTEGER *Input*

On entry: the number of boundary edges in the input mesh.

Constraint: NEDGE ≥ 1.

4: NNZMAX – INTEGER *Input*

On entry: the maximum number of nonzero entries in the matrix based on the input mesh. It is the dimension of the arrays IROW and ICOL as declared in the subroutine from which D06CCF is called.

Constraint: $4 \times \text{NELT} + \text{NV} \leq \text{NNZMAX} \leq \text{NV}^2$.

5: NNZ – INTEGER *Output*

On exit: the number of nonzero entries in the matrix based on the input mesh.

6: COOR$(2, \text{NV})$ – REAL (KIND=nag_wp) array *Input/Output*

On entry: COOR$(1, i)$ contains the x coordinate of the ith input mesh vertex, for $i = 1, 2, \ldots, \text{NV}$; while COOR$(2, i)$ contains the corresponding y coordinate.

On exit: COOR$(1, i)$ will contain the x coordinate of the ith renumbered mesh vertex, for $i = 1, 2, \ldots, \text{NV}$; while COOR$(2, i)$ will contain the corresponding y coordinate.

7: EDGE$(3, \text{NEDGE})$ – INTEGER array *Input/Output*

On entry: the specification of the boundary or interface edges. EDGE$(1, j)$ and EDGE$(2, j)$ contain the vertex numbers of the two end points of the jth boundary edge. EDGE$(3, j)$ is a user-supplied tag for the jth boundary or interface edge: EDGE$(3, j) = 0$ for an interior edge and has a nonzero tag otherwise.

Constraint: $1 \leq \text{EDGE}(i, j) \leq \text{NV}$ and $\text{EDGE}(1, j) \neq \text{EDGE}(2, j)$, for $i = 1, 2$ and $j = 1, 2, \ldots, \text{NEDGE}$.

On exit: the renumbered specification of the boundary or interface edges.

8: CONN$(3, \text{NELT})$ – INTEGER array *Input/Output*

On entry: the connectivity of the mesh between triangles and vertices. For each triangle j, CONN(i, j) gives the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \ldots, \text{NELT}$.

Constraint: $1 \leq \text{CONN}(i, j) \leq \text{NV}$ and $\text{CONN}(1, j) \neq \text{CONN}(2, j)$ and $\text{CONN}(1, j) \neq \text{CONN}(3, j)$ and $\text{CONN}(2, j) \neq \text{CONN}(3, j)$, for $i = 1, 2, 3$ and $j = 1, 2, \ldots, \text{NELT}$.

On exit: the renumbered connectivity of the mesh between triangles and vertices.

9: IROW(NNZMAX) – INTEGER array *Output*

10: ICOL(NNZMAX) – INTEGER array *Output*

On exit: the first NNZ elements contain the row and column indices of the nonzero elements supplied in the finite element matrix A.

11: ITRACE – INTEGER *Input*

On entry: the level of trace information required from D06CCF.

ITRACE ≤ 0
No output is generated.

ITRACE $= 1$
Information about the effect of the renumbering on the finite element matrix are output. This information includes the half bandwidth and the sparsity structure of this matrix before and after renumbering.

ITRACE > 1
The output is similar to that produced when ITRACE $= 1$ but the sparsities (for each row of the matrix, indices of nonzero entries) of the matrix before and after renumbering are also output.

12: IWORK(LIWORK) – INTEGER array *Workspace*
13: LIWORK – INTEGER *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D06CCF is called.

Constraint: LIWORK $\geq \max(\text{NNZMAX}, 20 \times \text{NV})$.

14: RWORK(LRWORK) – REAL (KIND=nag_wp) array *Workspace*
15: LRWORK – INTEGER *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D06CCF is called.

Constraint: LRWORK $\geq$ NV.

16: 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, NV < 3,
or NELT $> 2 \times$ NV $- 1$,
or NEDGE < 1,
or NNZMAX $< 4 \times$ NELT $+$ NV or NNZMAX $>$ NV^2
or CONN$(i, j) < 1$ or CONN$(i, j) >$ NV for some $i = 1, 2, 3$ and $j = 1, 2, \ldots,$ NELT,
or CONN$(1, j) =$ CONN$(2, j)$ or CONN$(1, j) =$ CONN$(3, j)$ or
CONN$(2, j) =$ CONN$(3, j)$ for some $j = 1, 2, \ldots,$ NELT,
or EDGE$(i, j) < 1$ or EDGE$(i, j) >$ NV for some $i = 1, 2$ and $j = 1, 2, \ldots,$ NEDGE,
or EDGE$(1, j) =$ EDGE$(2, j)$ for some $j = 1, 2, \ldots,$ NEDGE,
or LIWORK $< \max(\text{NNZMAX}, 20 \times \text{NV})$,
or LRWORK $<$ NV.

IFAIL $= 2$

A serious error has occurred during the computation of the compact sparsity of the finite element matrix or in an internal call to the renumbering routine. Check the input mesh, especially the connectivity between triangles and vertices (the argument CONN). If the problem persists, contact NAG.

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

Not applicable.

8 Parallelism and Performance

D06CCF is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

In this example, a geometry with two holes (two interior circles inside an exterior one) is considered. The geometry has been meshed using the simple incremental method (D06AAF) and it has 250 vertices and 402 triangles (see Figure 1 in Section 10.3). The routine D06BAF is used to renumber the vertices, and one can see the benefit in terms of the sparsity of the finite element matrix based on the renumbered mesh (see Figure 2 and 3 in Section 10.3).

10.1 Program Text

```
!   D06CCF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.
    Module d06ccfe_mod

!     D06CCF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp, x01aaf
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                            :: create_mesh
!     .. Parameters ..
      Integer, Parameter, Public        :: matout = 7, nout = 6, nvb1 = 40,     &
                                           nvb2 = 30, nvb3 = 30, nvmax = 260
      Integer, Parameter, Public        :: nvb = nvb1 + nvb2 + nvb3
      Integer, Parameter, Public        :: nedge = nvb
      Logical, Parameter, Public        :: pmesh = .False.

    Contains
      Subroutine create_mesh(edge,coor,nv,nelt,conn)

!       .. Use Statements ..
        Use nag_library, Only: d06aaf
```

```
!       .. Scalar Arguments ..
        Integer, Intent (Out)          :: nelt, nv
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: coor(2,nvmax)
        Integer, Intent (Out)          :: conn(3,2*nvmax-2), edge(3,nedge)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: coef, pi2, power, r, theta, theta_i, &
                                          x0, y0
        Integer                        :: i, ifail, itrace, liwork, lrwork
        Logical                        :: smooth
!       .. Local Arrays ..
        Real (Kind=nag_wp), Allocatable :: bspace(:), rwork(:)
        Integer, Allocatable           :: iwork(:)
!       .. Intrinsic Procedures ..
        Intrinsic                      :: cos, max, real, sin
!       .. Executable Statements ..

        lrwork = nvmax
        liwork = 16*nvmax + 2*nedge + max(4*nvmax+2,nedge-14)
        Allocate (bspace(nvb),rwork(lrwork),iwork(liwork))

!       Outer circle
        pi2 = 2.0_nag_wp*x01aaf(theta)
        theta = pi2/real(nvb1,kind=nag_wp)
        r = 1.0_nag_wp
        x0 = 0.0_nag_wp
        y0 = 0.0_nag_wp
        Do i = 1, nvb1
          theta_i = theta*real(i,kind=nag_wp)
          coor(1,i) = x0 + r*cos(theta_i)
          coor(2,i) = y0 + r*sin(theta_i)
        End Do
!       Larger inner circle
        theta = pi2/real(nvb2,kind=nag_wp)
        r = 0.49_nag_wp
        x0 = -0.5_nag_wp
        y0 = 0.0_nag_wp
        Do i = 1, nvb2
          theta_i = theta*real(i,kind=nag_wp)
          coor(1,nvb1+i) = x0 + r*cos(theta_i)
          coor(2,nvb1+i) = y0 + r*sin(theta_i)
        End Do
!       Smaller inner circle
        theta = pi2/real(nvb3,kind=nag_wp)
        r = 0.15_nag_wp
        x0 = -0.5_nag_wp
        y0 = 0.65_nag_wp
        Do i = 1, nvb3
          theta_i = theta*real(i,kind=nag_wp)
          coor(1,nvb1+nvb2+i) = x0 + r*cos(theta_i)
          coor(2,nvb1+nvb2+i) = y0 + r*sin(theta_i)
        End Do

!       Boundary edges
        Do i = 1, nedge
          edge(1,i) = i
          edge(2,i) = i + 1
          edge(3,i) = 1
        End Do
        edge(2,nvb1) = 1
        edge(2,nvb1+nvb2) = nvb1 + 1
        edge(2,nvb) = nvb1 + nvb2 + 1

!       Initialize mesh control parameters
        bspace(1:nvb) = 0.05E0_nag_wp
        smooth = .True.
        itrace = 0
        coef = 0.75E0_nag_wp
        power = 0.25E0_nag_wp

!       Call to the mesh generator
```

```
        ifail = 0
        Call d06aaf(nvb,nvmax,nedge,edge,nv,nelt,coor,conn,bspace,smooth,coef, &
          power,itrace,rwork,lrwork,iwork,liwork,ifail)

        Deallocate (bspace,rwork,iwork)

        Return
      End Subroutine create_mesh
    End Module d06ccfe_mod
    Program d06ccfe

!     D06CCF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d06cbf, d06ccf, nag_wp
      Use d06ccfe_mod, Only: create_mesh, matout, nedge, nout, nvmax, pmesh
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Integer                          :: i, ifail, itrace, liwork, lrwork,    &
                                          nelt, nnz, nnzmax, nv
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: coor(:,:), rwork(:)
      Integer, Allocatable             :: conn(:,:), edge(:,:), icol(:),       &
                                          irow(:), iwork(:)
!     .. Intrinsic Procedures ..
      Intrinsic                        :: max
!     .. Executable Statements ..
      Write (nout,*) 'D06CCF Example Program Results'
      Flush (nout)

!     Allocate arrays defining mesh
      Allocate (conn(3,2*nvmax-2),edge(3,nedge),coor(2,nvmax))

!     Define boundary mesh and Generate interior mesh
      Call create_mesh(edge,coor,nv,nelt,conn)

      nnzmax = nv**2
      liwork = max(nnzmax,20*nv)
      lrwork = nv
      Allocate (irow(nnzmax),icol(nnzmax),iwork(liwork),rwork(lrwork))

!     Compute the sparsity of the FE matrix
!     from the input geometry

      ifail = 0
      Call d06cbf(nv,nelt,nnzmax,conn,nnz,irow,icol,ifail)

      Write (nout,*)

      If (pmesh) Then

!       Output the sparsity of the mesh
        Write (matout,99998)(irow(i),icol(i),i=1,nnz)

      Else
        Write (nout,*) 'Matrix Sparsity characteristics before renumbering'
        Write (nout,99999) 'nv   =', nv
        Write (nout,99999) 'nnz  =', nnz
        Write (nout,99999) 'nelt =', nelt
      End If
      Flush (nout)

!     Call the renumbering routine and get the new sparsity

      itrace = 1

      ifail = 0
      Call d06ccf(nv,nelt,nedge,nnzmax,nnz,coor,edge,conn,irow,icol,itrace,    &
        iwork,liwork,rwork,lrwork,ifail)
```

```
      If (pmesh) Then

!       Output the sparsity of the renumbered mesh
        Write (matout,*)
        Write (matout,*)
        Write (matout,99998)(irow(i),icol(i),i=1,nnz)

!       Output the renumbered mesh
        Write (nout,99998) nv, nelt
        Write (nout,99997)(coor(1:2,i),i=1,nv)
        Write (nout,99996)(conn(1:3,i),i=1,nelt)

      Else
        Write (nout,*)
        Write (nout,*) 'Matrix Sparsity characteristics after renumbering'
        Write (nout,99999) 'nv   =', nv
        Write (nout,99999) 'nnz  =', nnz
        Write (nout,99999) 'nelt =', nelt
      End If

99999 Format (1X,A,I6)
99998 Format (1X,2I10)
99997 Format (2(2X,E13.6))
99996 Format (1X,3I10)
    End Program d06ccfe
```

10.2 Program Data

None.

10.3 Program Results

```
 D06CCF Example Program Results

 Matrix Sparsity characteristics before renumbering
 nv   =   260
 nnz  =  1626
 nelt =   422

 Initial half-bandwidth:   251    Initial profile: 19911
   Final half-bandwidth:    27      Final profile:  4012

 Matrix Sparsity characteristics after renumbering
 nv   =   260
 nnz  =  1626
 nelt =   422
```

Example Program

Figure 1: Mesh of the Geometry

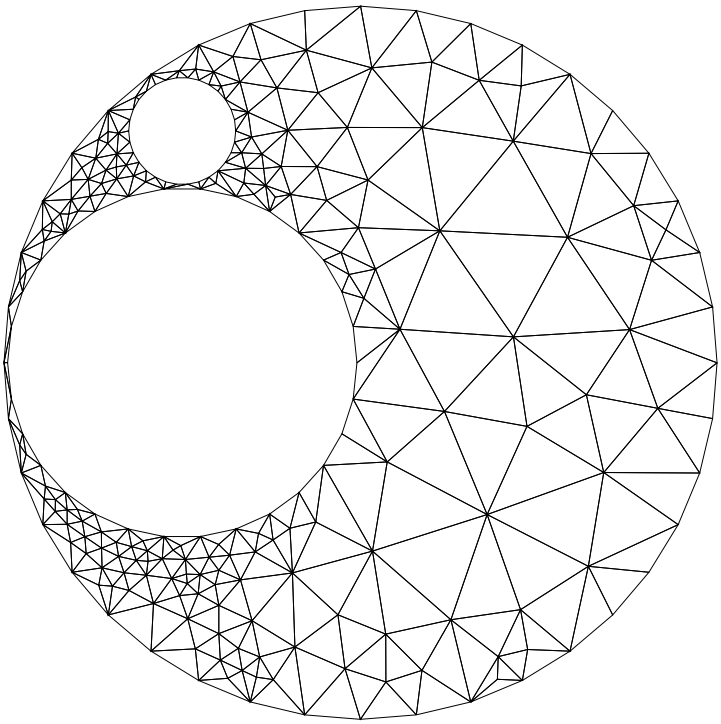

Figure 2: Sparsity of the FE Matrix Before Renumbering

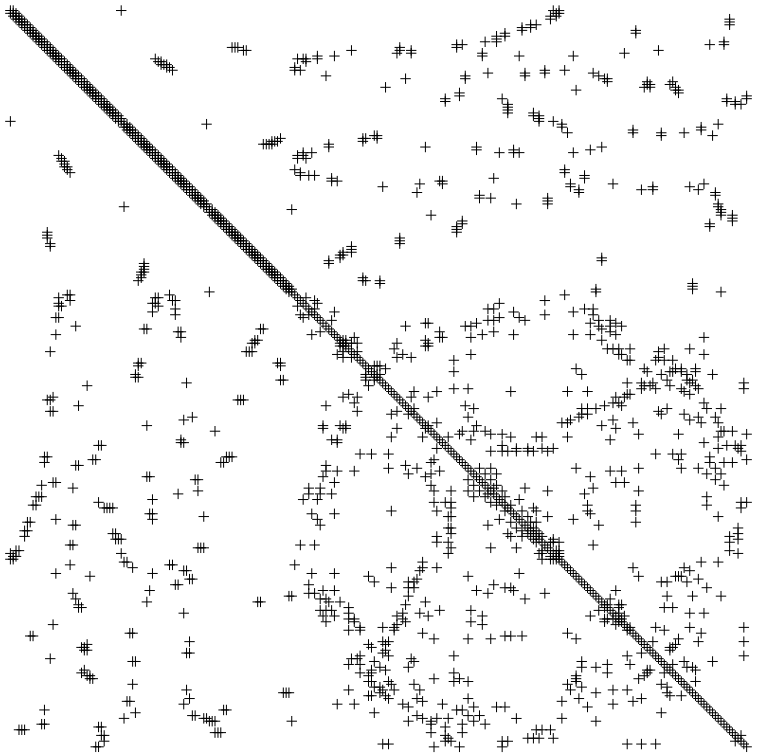

Figure 3: Sparsity of the FE Matrix After Renumbering