NAG C Library Function Document

nag_mesh2d_renum (d06ccc)

1 Purpose

nag_mesh2d_renum (d06ccc) 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

```
void nag_mesh2d_renum (Integer nv, Integer nelt, Integer nedge, Integer nnzmax,
     Integer *nnz, double coor[], Integer edge[], Integer conn[], Integer irow[],
     Integer icol[], Integer itrace, const char *outfile, NagError *fail)
```

3 Description

nag_mesh2d_renum (d06ccc) 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:

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

This function 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). nag_mesh2d_renum (d06ccc) also returns the sparsity structure of the matrix associated with the renumbered mesh.

This function 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 Parameters

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 the boundary edges in the input mesh.

Constraint: **nedge** ≥ 1.

4: **nnzmax** – Integer *Input*

On entry: the maximum number of non-zero entries in the matrix based on the input mesh. It is the dimension of the arrays **irow** and **icol** as declared in the function from which nag_mesh2d_renum (d06ccc) is called.

Constraint: $4 \times$ **nelt** $+$ **nv** $\leq$ **nnzmax** $\leq$ **nv**^{2}.

5: **nnz** – Integer * *Output*

On exit: the number of non-zero entries in the matrix based on the input mesh.

6: **coor**[$2 \times$ **nv**] – double *Input/Output*

Note: where **COOR**(i, j) appears in this document it refers to the array element **coor**$[2 \times (j-1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

On entry: **COOR**$(1, i)$ contains the x-coordinate of the ith input mesh vertex, for $i = 1, \ldots,$ **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 $i = 1, \ldots,$ **nv**; while **COOR**$(2, i)$ will contain the corresponding y-coordinate.

7: **edge**[$3 \times$ **nedge**] – Integer *Input/Output*

Note: where **EDGE**(i, j) appears in this document it refers to the array element **edge**$[3 \times (j-1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

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 non-zero tag otherwise. Note that the edge vertices are numbered from 1 to **nv**.

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

Constraint: $1 \leq$ **EDGE**$(i, j) \leq$ **nv** and **EDGE**$(1, j) \neq$ **EDGE**$(2, j)$ for $i = 1, 2$ and $j = 1, 2, \ldots,$ **nedge**.

8: **conn**[$3 \times$ **nelt**] – Integer *Input/Output*

Note: where **CONN**(i, j) appears in this document it refers to the array element **conn**$[3 \times (j-1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

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 $j = 1, \ldots,$ **nelt**. Note that the mesh vertices are numbered from 1 to **nv**.

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

Constraint: $1 \leq$ **CONN**$(i, j) \leq$ **nv** and **CONN**$(1, j) \neq$ **CONN**$(2, j)$ and **CONN**$(1, j) \neq$ **CONN**$(3, j)$ and **CONN**$(2, j) \neq$ **CONN**$(3, j)$ for $i = 1, 2, 3$ and $j = 1, 2, \ldots,$ **nelt**.

9: **irow**[**nnzmax**] – Integer *Output*
10: **icol**[**nnzmax**] – Integer *Output*

On exit: the first **nnz** elements contain the row and column indices of the non-zero elements supplied in the Finite Element matrix A.

11: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_renum (d06ccc) as follows:

if **itrace** ≤ 0, no output is generated;

if **itrace** $= 1$, then 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;

if **itrace** > 1, then the output is similar to that produced when **itrace** $= 1$ but the sparsities (for each row of the matrix, indices of non-zero entries) of the matrix before and after renumbering are also output.

12: **outfile** – char * *Input*

On entry: the name of a file to which diagnostic output will be directed. If **outfile** is NULL the diagnostic output will be directed to standard output.

13: **fail** – NagError * *Input/Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **nv** $= \langle value \rangle$.
Constraint: **nv** ≥ 3.

On entry, **nedge** $= \langle value \rangle$.
Constraint: **nedge** ≥ 1.

NE_INT_2

On entry, the endpoints of the edge j have the same index i: $j = \langle value \rangle$, $i = \langle value \rangle$.

On entry, **nelt** $= \langle value \rangle$, **nv** $= \langle value \rangle$.
Constraint: **nelt** $\leq 2 \times$ **nv** $- 1$.

On entry, vertices 2 and 3 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

On entry, vertices 1 and 3 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

On entry, vertices 1 and 2 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

NE_INT_3

On entry, **nnzmax** $< (4 \times$ **nelt** $+$ **nv**$)$ or **nnzmax** $>$ **nv**^{2}: **nnzmax** $= \langle value \rangle$, **nelt** $= \langle value \rangle$, **nv** $= \langle value \rangle$.

NE_INT_4

On entry, **edge**$(i, j) < 1$ or **edge**$(i, j) >$ **nv**, where **edge**(i, j) denotes **edge**$[3 \times (j - 1) + i - 1]$: **edge**$(i, j) = \langle value \rangle$, $i = \langle value \rangle$, $j = \langle value \rangle$, **nv** $= \langle value \rangle$.

On entry, **conn**$(i, j) < 1$ or **conn**$(i, j) >$ **nv**, where **conn**(i, j) denotes **conn**$[3 \times (j - 1) + i - 1]$: **conn**$(i, j) = \langle value \rangle$, $i = \langle value \rangle$, $j = \langle value \rangle$, **nv** $= \langle value \rangle$.

NE_FAIL_SPARSITY

An error has occurred during the computation of the compact sparsity of the {FE} matrix. Check the Triangle/Vertices connectivity.

NE_INTERNAL_ERROR

A serious error has occurred in an internal call to the renumbering routine. Check the input mesh especially the connectivity. Seek expert help.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

On entry, parameter $\langle value \rangle$ had an illegal value.

NE_NOT_WRITE_FILE

Cannot open file $\langle value \rangle$ for writing.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*value*⟩.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

7 Accuracy

Not applicable.

8 Further Comments

Not applicable.

9 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 (nag_mesh2d_inc (d06aac)) and it has 250 vertices and 402 triangles (see Figure 1). The function nag_mesh2d_bound (d06bac) 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).

9.1 Program Text

```
/* nag_mesh2d_renum (d06ccc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd06.h>

#define EDGE(I,J) edge[3*((J)-1)+(I)-1]
#define CONN(I,J) conn[3*((J)-1)+(I)-1]
#define COOR(I,J) coor[2*((J)-1)+(I)-1]

int main(int argc, char* argv[])
{
  Integer exit_status, i, itrace, nedge, nelt, nnz, nnzmax, nv, reftk;
  NagError fail;
  char pmesh[2];
  double *coor=0;
  Integer *conn=0, *edge=0, *icol=0, *irow=0;

  INIT_FAIL(fail);
  exit_status = 0;

  Vprintf(" d06ccc Example Program Results\n\n");

  /* Skip heading in data file */

  Vscanf("%*[^\n] ");

  /* Reading of the geometry */

  Vscanf("%ld", &nv);
  Vscanf("%ld", &nelt);
  Vscanf("%ld", &nedge);
  Vscanf("%*[^\n] ");
```

```
  nnzmax = 10*nv;

  /* Allocate memory */

  if ( !(coor = NAG_ALLOC(2*nv, double)) ||
       !(conn = NAG_ALLOC(3*nelt, Integer)) ||
       !(edge = NAG_ALLOC(3*nedge, Integer)) ||
       !(irow = NAG_ALLOC(nnzmax, Integer)) ||
       !(icol = NAG_ALLOC(nnzmax, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  for (i = 1; i <= nv; ++i)
    {
      Vscanf("%lf", &COOR(1,i));
      Vscanf("%lf", &COOR(2,i));
      Vscanf("%*[^\n] ");
    }

  for (i = 1; i <= nelt; ++i)
    {
      Vscanf("%ld", &CONN(1,i));
      Vscanf("%ld", &CONN(2,i));
      Vscanf("%ld", &CONN(3,i));
      Vscanf("%ld", &reftk);
      Vscanf("%*[^\n] ");
    }

  for (i = 1; i <= nedge; ++i)
    {
      Vscanf("%ld", &reftk);
      Vscanf("%ld", &EDGE(1,i));
      Vscanf("%ld", &EDGE(2,i));
      Vscanf("%ld", &EDGE(3,i));
      Vscanf("%*[^\n] ");
    }

  Vscanf(" ' %1s '", pmesh);
  Vscanf("%*[^\n] ");

  /* Compute the sparsity of the FE matrix */
  /* from the input geometry */

  d06cbc(nv, nelt, nnzmax, conn, &nnz, irow, icol, &fail);

  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {
          Vprintf(" The Matrix Sparsity characteristics\n");
          Vprintf(" before the renumbering\n");
          Vprintf(" nv  =%6ld\n", nv);
          Vprintf(" nnz =%6ld\n", nnz);
        }
      else if (pmesh[0] == 'Y')
        {
          /* Output the sparsity of the mesh to view */
          /* it using the NAG Graphics Library */

          Vprintf(" %10ld%10ld\n", nv, nnz);

          for (i = 0; i < nnz; ++i)
              Vprintf(" %10ld%10ld\n", irow[i], icol[i]);
        }
      else
        {
          Vprintf("Problem with the printing option Y or N\n");
```

```
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      Vprintf("Error from d06cbc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Call the renumbering routine and get the new sparsity */

  itrace = 1;

  d06ccc(nv, nelt, nedge, nnzmax, &nnz, coor, edge, conn, irow, icol,
         itrace, 0, &fail);

  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {
          Vprintf("\n The Matrix Sparsity characteristics\n");
          Vprintf(" after the renumbering\n");
          Vprintf(" nv   =%6ld\n", nv);
          Vprintf(" nnz  =%6ld\n", nnz);
          Vprintf(" nelt =%6ld\n", nelt);
        }
      else if (pmesh[0] == 'Y')
        {
          /* Output the sparsity of the renumbered mesh */
          /* to view it using the NAG Graphics Library */

          Vprintf("%10ld%10ld\n", nv, nnz);

          for (i = 0; i < nnz; ++i)
              Vprintf(" %10ld%10ld\n", irow[i], icol[i]);

          /* Output the renumbered mesh to view */
          /* it using the NAG Graphics Library */

          Vprintf(" %10ld%10ld\n", nv, nelt);

          for (i = 1; i <= nv; ++i)
              Vprintf("  %12.6e  %12.6e  \n",
                      COOR(1,i), COOR(2,i));

          reftk = 0;
          for (i = 1; i <= nelt; ++i)
              Vprintf(" %10ld%10ld%10ld%10ld\n",
                      CONN(1,i), CONN(2,i), CONN(3,i), reftk);
        }
    }
  else
    {
      Vprintf("Error from d06ccc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  if (coor) NAG_FREE(coor);
  if (conn) NAG_FREE(conn);
  if (edge) NAG_FREE(edge);
  if (irow) NAG_FREE(irow);
  if (icol) NAG_FREE(icol);

  return exit_status;
}
```

9.2 Program Data

Note: since the data file for this example is quite large only a section of it is reproduced in this document. The full data file is distributed with your implementation.

```
D06CCF Example Program Data
       250       402       100       :NV NELT NEDGE
  0.100000E+01  0.000000E+00
            .
            .
            .
  0.112781E+00  0.103479E+00    :COOR(1:2,1:NV)
        21        55        56         1
            .
            .
            .
       151       250       155         1   :(CONN(:,K), REFT, K=1,...,NELT)
   1   1   2   1
            .
            .
            .
 100 100  71   1   :(I1, EDGE(:,I), I=1,NEDGE)
'N'                :Printing option 'Y' or 'N'
```

9.3 Program Results

```
 d06ccc Example Program Results

 The Matrix Sparsity characteristics
 before the renumbering
 nv  =   250
 nnz =  1556

 INITIAL HALF-BAND-WIDTH:   234    INITIAL PROFILE: 18233
 FINAL HALF-BAND-WIDTH   :    28     FINAL PROFILE  :  4038

 The Matrix Sparsity characteristics
 after the renumbering
 nv   =   250
 nnz  =  1556
 nelt =   402
```

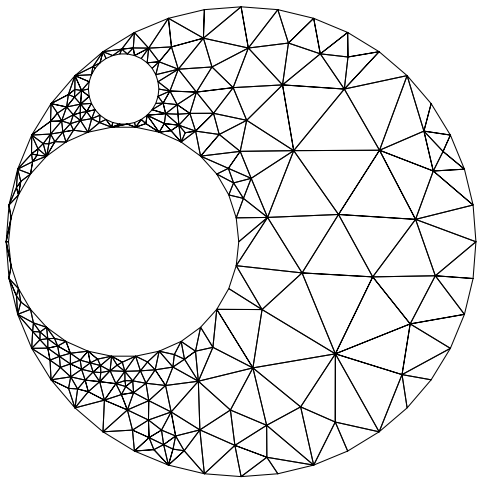

Figure 1
Mesh of the geometry

Figure 2
Sparsity of the matrix before (top) and after (bottom) the renumbering