NAG 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

```
#include <nag.h>
#include <nagd06.h>
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:

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

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 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 function from which nag_mesh2d_renum (d06ccc) is called.

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

5: **nnz** – Integer * *Output*

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

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

Note: the (i,j)th element of the matrix is stored in **coor**$[(j-1) \times 2 + i - 1]$.

On entry: **coor**$[(i-1) \times 2]$ contains the x coordinate of the ith input mesh vertex, for $i = 1, 2, \ldots, \mathbf{nv}$; while **coor**$[(i-1) \times 2 + 1]$ contains the corresponding y coordinate.

On exit: **coor**$[(i-1) \times 2]$ will contain the x coordinate of the ith renumbered mesh vertex, for $i = 1, 2, \ldots, \mathbf{nv}$; while **coor**$[(i-1) \times 2 + 1]$ will contain the corresponding y coordinate.

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

Note: the (i,j)th element of the matrix is stored in **edge**$[(j-1) \times 3 + i - 1]$.

On entry: the specification of the boundary or interface edges. **edge**$[(j-1) \times 3]$ and **edge**$[(j-1) \times 3 + 1]$ contain the vertex numbers of the two end points of the jth boundary edge. **edge**$[(j-1) \times 3 + 2]$ is a user-supplied tag for the jth boundary or interface edge: **edge**$[(j-1) \times 3 + 2] = 0$ for an interior edge and has a nonzero tag otherwise. Note that the edge vertices are numbered from 1 to **nv**.

Constraint: $1 \leq \mathbf{edge}[(j-1) \times 3 + i - 1] \leq \mathbf{nv}$ and $\mathbf{edge}[(j-1) \times 3] \neq \mathbf{edge}[(j-1) \times 3 + 1]$, for $i = 1, 2$ and $j = 1, 2, \ldots, \mathbf{nedge}$.

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

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

Note: the (i,j)th element of the matrix is stored in **conn**$[(j-1) \times 3 + i - 1]$.

On entry: the connectivity of the mesh between triangles and vertices. For each triangle j, **conn**$[(j-1) \times 3 + i - 1]$ gives the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \ldots, \mathbf{nelt}$. Note that the mesh vertices are numbered from 1 to **nv**.

Constraint: $1 \leq \mathbf{conn}[(j-1) \times 3 + i - 1] \leq \mathbf{nv}$ and $\mathbf{conn}[(j-1) \times 3] \neq \mathbf{conn}[(j-1) \times 3 + 1]$ and $\mathbf{conn}[(j-1) \times 3] \neq \mathbf{conn}[(j-1) \times 3 + 2]$ and
$\mathbf{conn}[(j-1) \times 3 + 1] \neq \mathbf{conn}[(j-1) \times 3 + 2]$, for $i = 1, 2, 3$ and $j = 1, 2, \ldots, \mathbf{nelt}$.

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

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

10: **icol**[**nnzmax**] – Integer *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 nag_mesh2d_renum (d06ccc).

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: **outfile** – const 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 argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_FAIL_SPARSITY

An error has occurred during the computation of the compact sparsity of the finite element matrix. Check the Triangle/Vertices connectivity.

NE_INT

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

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

NE_INT_2

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

On entry, the end points of the edge J have the same index I: $J = \langle value \rangle$ and $I = \langle value \rangle$.

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

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

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

NE_INT_3

On entry, **nnzmax** $= \langle value \rangle$, **nelt** $= \langle value \rangle$ and **nv** $= \langle value \rangle$.
Constraint: **nnzmax** $\geq (4 \times$ **nelt** $+$ **nv**$)$ and **nnzmax** $\leq$ **nv**^{2}.

NE_INT_4

On entry, $\mathbf{CONN}(I,J) = \langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{nv} = \langle value \rangle$.
Constraint: $\mathbf{CONN}(I,J) \geq 1$ and $\mathbf{CONN}(I,J) \leq \mathbf{nv}$, where $\mathbf{CONN}(I,J)$ denotes $\mathbf{conn}[(J-1) \times 3 + I - 1]$.

On entry, $\mathbf{EDGE}(I,J) = \langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{nv} = \langle value \rangle$.
Constraint: $\mathbf{EDGE}(I,J) \geq 1$ and $\mathbf{EDGE}(I,J) \leq \mathbf{nv}$, where $\mathbf{EDGE}(I,J)$ denotes $\mathbf{edge}[(J-1) \times 3 + I - 1]$.

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 contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

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

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_NOT_CLOSE_FILE

Cannot close file $\langle value \rangle$.

NE_NOT_WRITE_FILE

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mesh2d_renum (d06ccc) 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 function. 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 (nag_mesh2d_inc (d06aac)) and it has 250 vertices and 402 triangles (see Figure 1 in Section 10.3). 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 and 3 in Section 10.3).

10.1 Program Text

```
/* nag_mesh2d_renum (d06ccc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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(void)
{
  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;

  printf(" nag_mesh2d_renum (d06ccc) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */

#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Reading of the geometry */

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &nv);
#else
  scanf("%" NAG_IFMT "", &nv);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &nelt);
#else
  scanf("%" NAG_IFMT "", &nelt);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "", &nedge);
#else
  scanf("%" NAG_IFMT "", &nedge);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  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)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  for (i = 1; i <= nv; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &COOR(1, i));
#else
    scanf("%lf", &COOR(1, i));
#endif
#ifdef _WIN32
    scanf_s("%lf", &COOR(2, i));
#else
    scanf("%lf", &COOR(2, i));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  for (i = 1; i <= nelt; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &CONN(1, i));
#else
    scanf("%" NAG_IFMT "", &CONN(1, i));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &CONN(2, i));
#else
    scanf("%" NAG_IFMT "", &CONN(2, i));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &CONN(3, i));
#else
    scanf("%" NAG_IFMT "", &CONN(3, i));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &reftk);
#else
    scanf("%" NAG_IFMT "", &reftk);
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

  for (i = 1; i <= nedge; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &reftk);
#else
    scanf("%" NAG_IFMT "", &reftk);
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &EDGE(1, i));
#else
    scanf("%" NAG_IFMT "", &EDGE(1, i));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &EDGE(2, i));
#else
    scanf("%" NAG_IFMT "", &EDGE(2, i));
```

```
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &EDGE(3, i));
#else
    scanf("%" NAG_IFMT "", &EDGE(3, i));
#endif
#ifdef _WIN32
    scanf_s("%*[^\n] ");
#else
    scanf("%*[^\n] ");
#endif
  }

#ifdef _WIN32
  scanf_s(" ' %1s '", pmesh, (unsigned)_countof(pmesh));
#else
  scanf(" ' %1s '", pmesh);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

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

  /* nag_mesh2d_sparse (d06cbc).
   * Generates a sparsity pattern of a Finite Element matrix
   * associated with a given mesh
   */
  nag_mesh2d_sparse(nv, nelt, nnzmax, conn, &nnz, irow, icol, &fail);

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

      printf(" %10" NAG_IFMT "%10" NAG_IFMT "\n", nv, nnz);

      for (i = 0; i < nnz; ++i)
        printf(" %10" NAG_IFMT "%10" NAG_IFMT "\n", irow[i], icol[i]);
    }
    else {
      printf("Problem with the printing option Y or N\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Error from nag_mesh2d_sparse (d06cbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

  itrace = 1;

  /* nag_mesh2d_renum (d06ccc).
   * Renumbers a given mesh using Gibbs method
   */
  fflush(stdout);
  nag_mesh2d_renum(nv, nelt, nedge, nnzmax, &nnz, coor, edge, conn, irow,
                   icol, itrace, 0, &fail);
```

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

      printf("%10" NAG_IFMT "%10" NAG_IFMT "\n", nv, nnz);

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

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

      printf(" %10" NAG_IFMT "%10" NAG_IFMT "\n", nv, nelt);

      for (i = 1; i <= nv; ++i)
        printf("  %15.6e  %15.6e  \n", COOR(1, i), COOR(2, i));

      reftk = 0;
      for (i = 1; i <= nelt; ++i)
        printf(" %10" NAG_IFMT "%10" NAG_IFMT "%10" NAG_IFMT "%10" NAG_IFMT
               "\n", CONN(1, i), CONN(2, i), CONN(3, i), reftk);
    }
  }
  else {
    printf("Error from nag_mesh2d_renum (d06ccc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

  return exit_status;
}
```

10.2 Program Data

```
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'
```

10.3 Program Results

```
nag_mesh2d_renum (d06ccc) Example Program Results

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

Initial half-bandwidth:   234    Initial profile: 18233
  Final half-bandwidth:    28      Final profile:  4038

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

Example Program
Figure 1: Mesh of the Geometry

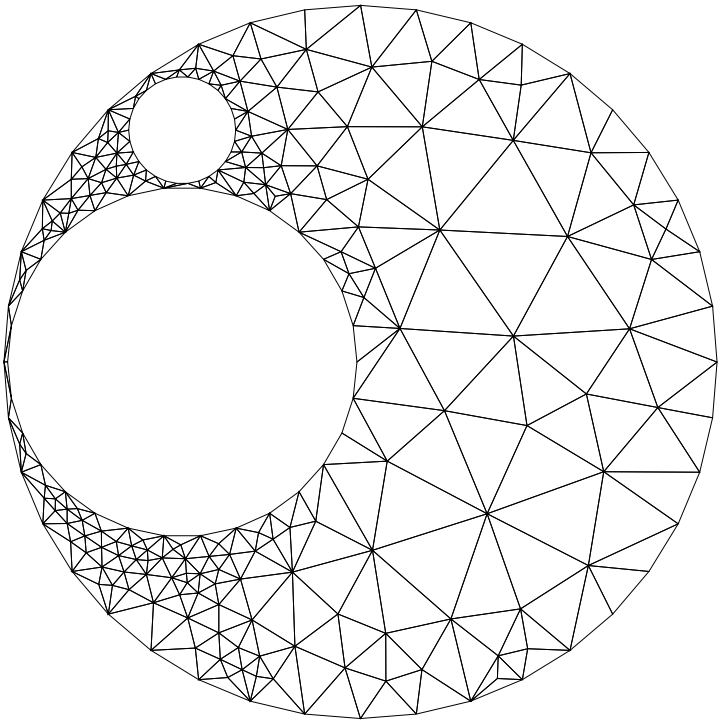

Figure 2: Sparsity of the FE Matrix Before Renumbering

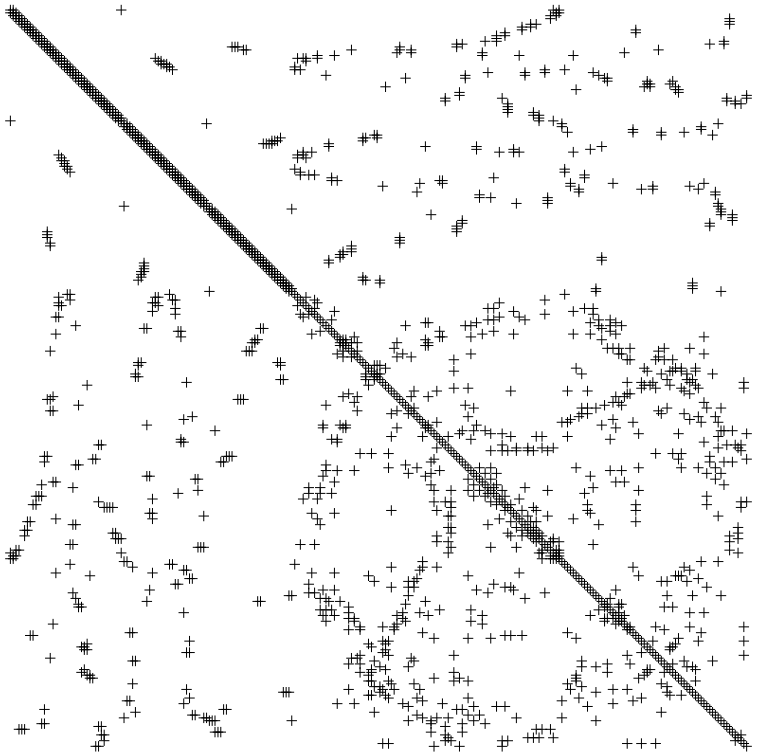

Figure 3: Sparsity of the FE Matrix After Renumbering