NAG Library Function Document

nag_mesh2d_inc (d06aac)

1 Purpose

nag_mesh2d_inc (d06aac) generates a triangular mesh of a closed polygonal region in $\mathbb{R}^2$, given a mesh of its boundary. It uses a simple incremental method.

2 Specification

```
#include <nag.h>
#include <nagd06.h>
void nag_mesh2d_inc (Integer nvb, Integer nvmax, Integer nedge,
     const Integer edge[], Integer *nv, Integer *nelt, double coor[],
     Integer conn[], const double bspace[], Nag_Boolean smooth, double coef,
     double power, Integer itrace, const char *outfile, NagError *fail)
```

3 Description

nag_mesh2d_inc (d06aac) generates the set of interior vertices using a process based on a simple incremental method. A smoothing of the mesh is optionally available. For more details about the triangulation method, consult the d06 Chapter Introduction as well as George and Borouchaki (1998).

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

4 References

George P L and Borouchaki H (1998) *Delaunay Triangulation and Meshing: Application to Finite Elements* Editions HERMES, Paris

5 Arguments

1: **nvb** – Integer *Input*

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

Constraint: $3 \leq$ **nvb** $\leq$ **nvmax**.

2: **nvmax** – Integer *Input*

On entry: the maximum number of vertices in the mesh to be generated.

3: **nedge** – Integer *Input*

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

Constraint: **nedge** ≥ 1.

4: **edge**[**3** $\times$ **nedge**] – const Integer *Input*

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 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 edge and is not used by nag_mesh2d_inc (d06aac). Note that the edge vertices are numbered from 1 to **nvb**.

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

5: **nv** – Integer * *Output*

On exit: the total number of vertices in the output mesh (including both boundary and interior vertices). If $\mathbf{nvb} = \mathbf{nvmax}$, no interior vertices will be generated and $\mathbf{nv} = \mathbf{nvb}$.

6: **nelt** – Integer * *Output*

On exit: the number of triangular elements in the mesh.

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

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

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

On exit: $\mathbf{coor}[(i-1) \times 2]$ will contain the x coordinate of the $(i - \mathbf{nvb})$th generated interior mesh vertex; while $\mathbf{coor}[(i-1) \times 2 + 1]$ will contain the corresponding y coordinate, for $i = \mathbf{nvb} + 1, \ldots, \mathbf{nv}$. The remaining elements are unchanged.

8: **conn**[$\mathbf{3} \times \mathbf{2} \times (\mathbf{nvmax} - \mathbf{1})$] – Integer *Output*

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

On exit: the connectivity of the mesh between triangles and vertices. For each triangle j, $\mathbf{conn}[(j-1) \times 3 + i - 1]$ 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**.

9: **bspace**[**nvb**] – const double *Input*

On entry: the desired mesh spacing (triangle diameter, which is the length of the longer edge of the triangle) near the boundary vertices.

Constraint: $\mathbf{bspace}[i-1] > 0.0$, for $i = 1, 2, \ldots, \mathbf{nvb}$.

10: **smooth** – Nag_Boolean *Input*

On entry: indicates whether or not mesh smoothing should be performed.

If **smooth** = Nag_TRUE, the smoothing is performed; otherwise no smoothing is performed.

11: **coef** – double *Input*

On entry: the coefficient in the stopping criteria for the generation of interior vertices. This argument controls the triangle density and the number of triangles generated is in $O(\mathbf{coef}^2)$. The mesh will be finer if **coef** is greater than 0.7165 and 0.75 is a good value.

Suggested value: 0.75.

12: **power** – double *Input*

On entry: controls the rate of change of the mesh size during the generation of interior vertices. The smaller the value of **power**, the faster the decrease in element size away from the boundary.

Suggested value: 0.25.

Constraint: $0.1 \leq \mathbf{power} \leq 10.0$.

13: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_inc (d06aac).

itrace ≤ 0
No output is generated.

itrace ≥ 1
Output from the meshing solver is printed. This output contains details of the vertices and triangles generated by the process.

You are advised to set **itrace** $= 0$, unless you are experienced with finite element mesh generation.

14: **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.

15: **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_INT

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

NE_INT_2

On entry, **nvb** $= \langle value \rangle$ and **nvmax** $= \langle value \rangle$.
Constraint: $3 \leq$ **nvb** $\leq$ **nvmax**.

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

NE_INT_4

On entry, **EDGE**$(I, J) = \langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and **nvb** $= \langle value \rangle$.
Constraint: **EDGE**$(I, J) \geq 1$ and **EDGE**$(I, J) \leq$ **nvb**, where **EDGE**(I, J) denotes **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.

NE_MESH_ERROR

An error has occurred during the generation of the interior mesh. Check the inputs of the boundary.

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.

NE_REAL

On entry, **power** $= \langle value \rangle$.
Constraint: **power** ≤ 10.0.

On entry, **power** $= \langle value \rangle$.
Constraint: **power** ≥ 0.1.

NE_REAL_ARRAY_INPUT

On entry, **bspace**$[I-1] = \langle value \rangle$ and $I = \langle value \rangle$.
Constraint: **bspace**$[I-1] > 0.0$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mesh2d_inc (d06aac) is not threaded in any implementation.

9 Further Comments

The position of the internal vertices is a function of the positions of the vertices on the given boundary. A fine mesh on the boundary results in a fine mesh in the interior. The algorithm allows you to obtain a denser interior mesh by varying **nvmax**, **bspace**, **coef** and **power**. But you are advised to manipulate the last two arguments with care.

You are advised to take care to set the boundary inputs properly, especially for a boundary with multiply connected components. The orientation of the interior boundaries should be in **clockwise** order and opposite to that of the exterior boundary. If the boundary has only one connected component, its orientation should be **anticlockwise**.

10 Example

In this example, a geometry with two holes (two interior circles inside an exterior one) is meshed using the simple incremental method (see the d06 Chapter Introduction). The exterior circle is centred at the origin with a radius 1.0, the first interior circle is centred at the point $(-0.5, 0.0)$ with a radius 0.49, and the second one is centred at the point $(-0.5, 0.65)$ with a radius 0.15. Note that the points $(-1.0, 0.0)$ and $(-0.5, 0.5)$) are points of 'near tangency' between the exterior circle and the first and second circles.

The boundary mesh has 100 vertices and 100 edges (see Figure 1 in Section 10.3). Note that the particular mesh generated could be sensitive to the ***machine precision*** and therefore may differ from one implementation to another. Figure 2 in Section 10.3 contains the output mesh.

10.1 Program Text

```
/* nag_mesh2d_inc (d06aac) 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)
{
  const Integer nvmax = 250;
  double coef, power;
  Integer exit_status, i1, i, itrace, k, nedge, nelt, nv, nvb, reftk;
  Nag_Boolean smooth;
  NagError fail;
  char pmesh[2];
  double *bspace = 0, *coor = 0;
  Integer *conn = 0, *edge = 0;

  INIT_FAIL(fail);

  exit_status = 0;

  printf("nag_mesh2d_inc (d06aac) Example Program Results\n\n");

  /* Skip heading in data file */

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

  /* Reading of the geometry */

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

  if (nvb > nvmax) {
    printf("Problem with the array dimensions\n");
    printf(" nvb nvmax %6" NAG_IFMT "%6" NAG_IFMT "\n", nvb, nvmax);
    printf(" Please increase the value of nvmax\n");
    exit_status = -1;
    goto END;
  }

  /* Allocate memory */

  if (!(bspace = NAG_ALLOC(nvb, double)) ||
      !(coor = NAG_ALLOC(2 * nvmax, double)) ||
```

```
      !(conn = NAG_ALLOC(3 * (2 * nvmax - 1), Integer)) ||
      !(edge = NAG_ALLOC(3 * nedge, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Coordinates of the boundary mesh vertices and boundary edges */

  for (i = 1; i <= nvb; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &i1);
#else
    scanf("%" NAG_IFMT "", &i1);
#endif
#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 <= nedge; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &i1);
#else
    scanf("%" NAG_IFMT "", &i1);
#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 '%*[^\n]", pmesh, (unsigned)_countof(pmesh));
#else
  scanf(" ' %1s '%*[^\n]", pmesh);
#endif

  /* Initialize mesh control parameters */

  for (i = 0; i < nvb; ++i)
    bspace[i] = 0.05;
```

```
  smooth = Nag_TRUE;
  itrace = 0;
  coef = 0.75;
  power = 0.25;

  /* Call to the mesh generator */

  /* nag_mesh2d_inc (d06aac).
   * Generates a two-dimensional mesh using a simple
   * incremental method
   */
  nag_mesh2d_inc(nvb, nvmax, nedge, edge, &nv, &nelt, coor, conn, bspace,
                 smooth, coef, power, itrace, 0, &fail);

  if (fail.code == NE_NOERROR) {
    if (pmesh[0] == 'N') {
      printf(" nv   =%6" NAG_IFMT "\n", nv);
      printf(" nelt =%6" NAG_IFMT "\n", nelt);
    }
    else if (pmesh[0] == 'Y') {
      /* Output the 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 (k = 1; k <= nelt; ++k) {
        printf(" %10" NAG_IFMT "%10" NAG_IFMT "%10" NAG_IFMT "%10" NAG_IFMT
               "\n", CONN(1, k), CONN(2, k), CONN(3, k), reftk);
      }
    }
    else {
      printf("Problem with the printing option Y or N\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Error from nag_mesh2d_inc (d06aac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

END:
  NAG_FREE(bspace);
  NAG_FREE(coor);
  NAG_FREE(conn);
  NAG_FREE(edge);

  return exit_status;
}
```

10.2 Program Data

Note 1: 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.

```
D06AAF Example Program Data
       100       100     :NVB NEDGE
     1  0.100000E+01  0.000000E+00
              .
              .
              .
   100  -.353278E+00  0.681187E+00 :(I1, COOR(:,I),I=1,...,NVB)
   1   1   2   1
              .
```

```
            .
            .
  99  99 100   1
 100 100  71   1  :(I1, EDGE(:,I), I=1,...,NEDGE)
'N'              :Printing option 'Y' or 'N'
```

10.3 Program Results

```
nag_mesh2d_inc (d06aac) Example Program Results

 nv   =   250
 nelt =   402
```

Example Program
Figure 1: The Geometry of Circular Region With Two Holes

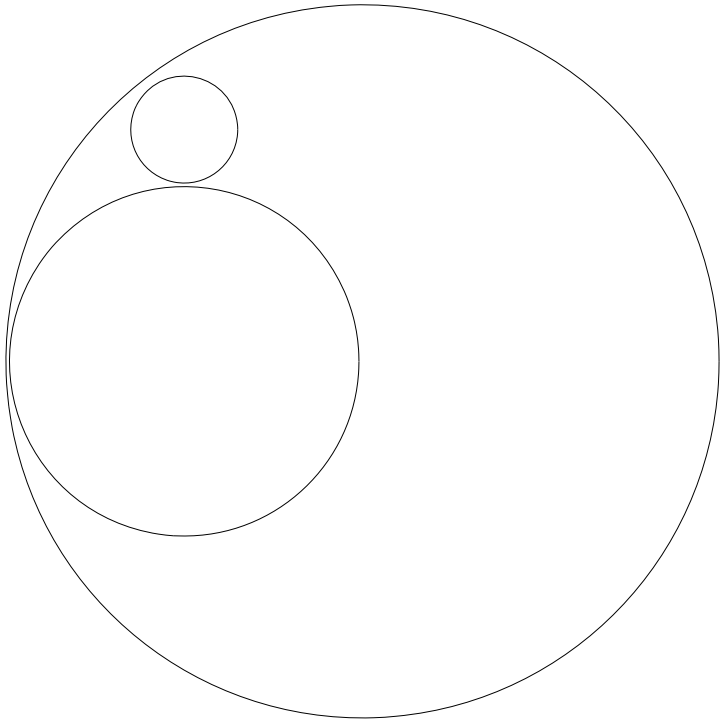

Figure 2: Mesh Generated on the Geometry With Two Holes