NAG Library Function Document

nag_mesh2d_delaunay (d06abc)

1 Purpose

nag_mesh2d_delaunay (d06abc) generates a triangular mesh of a closed polygonal region in $\mathbb{R}^2$, given a mesh of its boundary. It uses a Delaunay–Voronoi process, based on an incremental method.

2 Specification

```
#include <nag.h>
#include <nagd06.h>
void nag_mesh2d_delaunay (Integer nvb, Integer nvint, Integer nvmax,
     Integer nedge, const Integer edge[], Integer *nv, Integer *nelt,
     double coor[], Integer conn[], const double weight[], Integer npropa,
     Integer itrace, const char *outfile, NagError *fail)
```

3 Description

nag_mesh2d_delaunay (d06abc) generates the set of interior vertices using a Delaunay–Voronoi process, based on an incremental method. It allows you to specify a number of fixed interior mesh vertices together with weights which allow concentration of the mesh in their neighbourhood. 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: **nvb** ≥ 3.

2: **nvint** – Integer *Input*

On entry: the number of fixed interior mesh vertices to which a weight will be applied.

Constraint: **nvint** ≥ 0.

3: **nvmax** – Integer *Input*

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

Constraint: **nvmax** $\geq$ **nvb** $+$ **nvint**.

4: **nedge** – Integer *Input*

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

Constraint: **nedge** ≥ 1.

5: **edge**[**3** × **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_delaunay (d06abc). Note that the edge vertices are numbered from 1 to **nvb**.

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

6: **nv** – Integer * *Output*

On exit: the total number of vertices in the output mesh (including both boundary and interior vertices). If **nvb** + **nvint** = **nvmax**, no interior vertices will be generated and **nv** = **nvmax**.

7: **nelt** – Integer * *Output*

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

8: **coor**[**2** × **nvmax**] – 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 boundary mesh vertex, for $i = 1,2,\ldots,$ **nvb**. **coor**$[(i-1)\times 2]$ contains the x coordinate of the $(i-$ **nvb**$)$th fixed interior vertex, for $i =$ **nvb** $+1,\ldots,$ **nvb** + **nvint**. For boundary and interior vertices, **coor**$[(i-1)\times 2+1]$ contains the corresponding y coordinate, for $i = 1,2,\ldots,$ **nvb** + **nvint**.

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

9: **conn**[**3** × (**2** × **nvmax** + **5**)] – Integer *Output*

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

On exit: 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 $j = 1,2,\ldots,$ **nelt**. Note that the mesh vertices are numbered from 1 to **nv**.

10: **weight**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **weight** must be at least $\max(1,$ **nvint**$)$.

On entry: the weight of fixed interior vertices. It is the diameter of triangles (length of the longer edge) created around each of the given interior vertices.

Constraint: if **nvint** > 0, **weight**$[i-1] > 0.0$, for $i = 1,2,\ldots,$ **nvint**.

11: **npropa** – Integer *Input*

On entry: the propagation type and coefficient, the argument **npropa** is used when the internal points are created. They are distributed in a geometric manner if **npropa** is positive and in an arithmetic manner if it is negative. For more details see Section 9.

Constraint: **npropa** $\neq 0$.

12: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_delaunay (d06abc).

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.

13: **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.

14: **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** $=$ ⟨*value*⟩.
Constraint: **nedge** ≥ 1.

On entry, **npropa** $= 0$.

On entry, **nvb** $=$ ⟨*value*⟩.
Constraint: **nvb** ≥ 3.

On entry, **nvint** $=$ ⟨*value*⟩.
Constraint: **nvint** ≥ 0.

NE_INT_2

On entry, the end points of the edge J have the same index I: $J =$ ⟨*value*⟩ and $I =$ ⟨*value*⟩.

NE_INT_3

On entry, **nvb** $=$ ⟨*value*⟩, **nvint** $=$ ⟨*value*⟩ and **nvmax** $=$ ⟨*value*⟩.
Constraint: **nvmax** $\geq$ **nvb** $+$ **nvint**.

NE_INT_4

On entry, **EDGE**$(I, J) =$ ⟨*value*⟩, $I =$ ⟨*value*⟩, $J =$ ⟨*value*⟩ and **nvb** $=$ ⟨*value*⟩.
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 boundary mesh. It appears that **nvmax** is not large enough: **nvmax** $= \langle value \rangle$.

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_ARRAY_INPUT

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mesh2d_delaunay (d06abc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

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

The position of the internal vertices is a function position of the vertices on the given boundary. A fine mesh on the boundary results in a fine mesh in the interior. To dilute the influence of the data on the interior of the domain, the value of **npropa** can be changed. The propagation coefficient is calculated as: $\omega = 1 + \dfrac{a - 1.0}{20.0}$, where a is the absolute value of **npropa**. During the process vertices are generated on edges of the mesh $\mathcal{T}_i$ to obtain the mesh $\mathcal{T}_{i+1}$ in the general incremental method (consult the d06 Chapter Introduction or George and Borouchaki (1998)). This generation uses the coefficient ω, and it is geometric if **npropa** > 0, and arithmetic otherwise. But increasing the value of a may lead to failure of the process, due to precision, especially in geometries with holes. So you are advised to manipulate the argument **npropa** 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 wings inside an exterior circle) is meshed using a Delaunay–Voronoi method. The exterior circle is centred at the point $(1.0, 0.0)$ with a radius 3. The main wing, using aerofoil RAE 2822 data, lies between the origin and the centre of the circle, while the secondary aerofoil is produced from the first by performing a translation, a scale reduction and a rotation. To be able to carry out some realistic computation on that geometry, some interior points have been introduced to have a finer mesh in the wake of those aerofoils.

The boundary mesh has 296 vertices and 296 edges (see Section 10.3 top). Note that the particular mesh generated could be sensitive to the ***machine precision*** and therefore may differ from one implementation to another. The interior meshes for different values of **npropa** are given in Section 10.3.

10.1 Program Text

```
/* nag_mesh2d_delaunay (d06abc) 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 = 6000, nvint = 40;
  double dnvint;
  Integer exit_status, i, itrace, j, k, nedge, nelt,
    npropa, nv, nvb, reftk, i1, nearest, nv_near, nelt_near;
  NagError fail;
  char pmesh[2];
  double *coor = 0, *weight = 0;
  Integer *conn = 0, *edge = 0;

  INIT_FAIL(fail);

  exit_status = 0;

  printf("nag_mesh2d_delaunay (d06abc) 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 (!(coor = NAG_ALLOC(2 * nvmax, double)) ||
      !(weight = NAG_ALLOC(nvint, double)) ||
      !(conn = NAG_ALLOC(3 * (2 * nvmax + 5), 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, i1));
#else
    scanf("%lf", &COOR(1, i1));
#endif
#ifdef _WIN32
    scanf_s("%lf", &COOR(2, i1));
#else
    scanf("%lf", &COOR(2, i1));
#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, i1));
#else
    scanf("%" NAG_IFMT "", &EDGE(1, i1));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &EDGE(2, i1));
#else
    scanf("%" NAG_IFMT "", &EDGE(2, i1));
#endif
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &EDGE(3, i1));
#else
    scanf("%" NAG_IFMT "", &EDGE(3, i1));
#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 */

  itrace = 0;

  /* Generation of interior vertices on the RAE airfoils wake */

  dnvint = 2.5 / (double) (nvint + 1);

  for (i = 1; i <= nvint; ++i) {
    i1 = nvb + i;
    COOR(1, i1) = (double) i *dnvint + 1.38;
    COOR(2, i1) = -0.27 * COOR(1, i1) + 0.2;
    weight[i - 1] = 0.01;
  }

  /* Loop on the propagation coef */

  nearest = 250;
  for (j = 0; j < 4; ++j) {
    switch (j) {
    case 0:
      npropa = -5;
      break;
    case 1:
      npropa = -1;
      break;
    case 2:
      npropa = 1;
      break;
    default:
      npropa = 5;
    }

    /* Call to the 2D Delaunay-Voronoi mesh generator */

    /* nag_mesh2d_delaunay (d06abc).
     * Generates a two-dimensional mesh using a Delaunay-Voronoi
     * process
     */
    nag_mesh2d_delaunay(nvb, nvint, nvmax, nedge, edge, &nv, &nelt, coor,
                        conn, weight, npropa, itrace, 0, &fail);

    if (fail.code == NE_NOERROR) {
      if (pmesh[0] == 'N') {
        printf(" Mesh characteristics with npropa =%6" NAG_IFMT "\n", npropa);
        nv_near = ((nv+nearest/2)/nearest)*nearest;
        nelt_near = ((nelt+nearest/2)/nearest)*nearest;
        printf(" nv   =%10" NAG_IFMT " to the nearest %3" NAG_IFMT "\n",
               nv_near, nearest);
        printf(" nelt =%10" NAG_IFMT " to the nearest %3" NAG_IFMT "\n",
               nelt_near, nearest);
      }
      else if (pmesh[0] == 'Y') {
        /* Output the mesh in a form suitable for printing */

        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_delaunay (d06abc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  }

END:
  NAG_FREE(coor);
  NAG_FREE(weight);
  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.

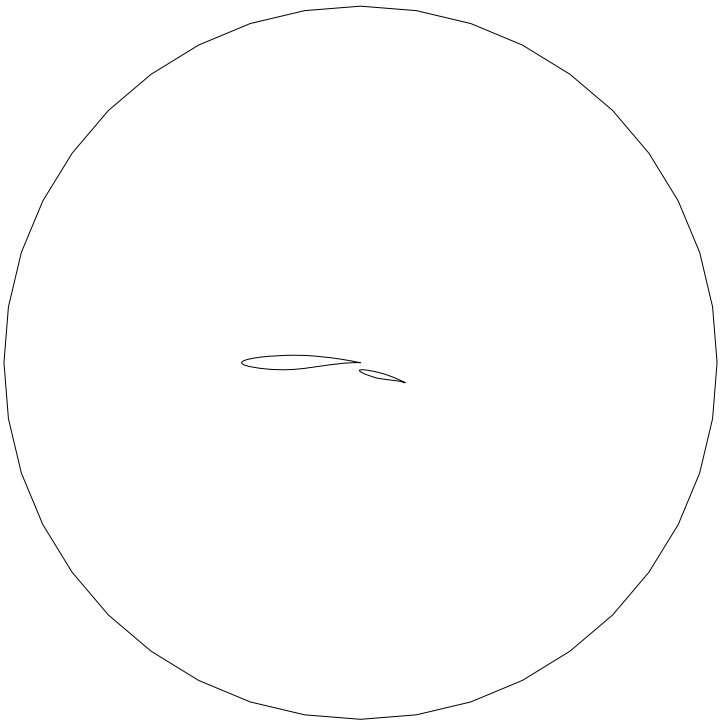
```
D06ABF Example Program Data
       296       296     :NVB NEDGE
     1  0.400000E+01  0.000000E+00
              .
              .
              .
   296  0.991387E+00  -.659880E-01 :(I1, COOR(:,I),I=1,...,NVB)
   1   1   2   0
              .
              .
              .
 296 296 169   0  :(I1, EDGE(:,I), I=1,...,NEDGE)
'N'               :Printing option 'Y' or 'N'
```

10.3 Program Results

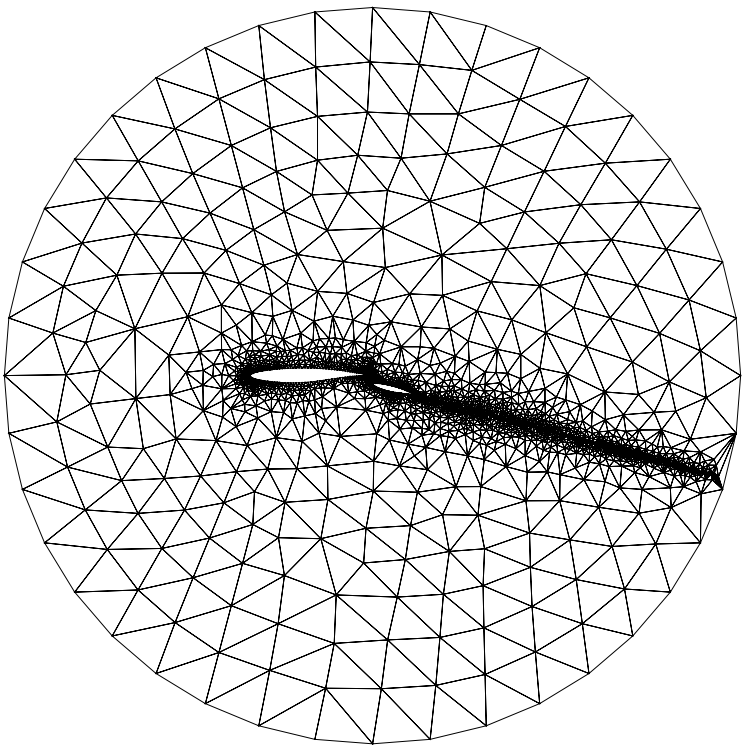
```
nag_mesh2d_delaunay (d06abc) Example Program Results

 Mesh characteristics with npropa =    -5
 nv   =      2250 to the nearest 250
 nelt =      4250 to the nearest 250
 Mesh characteristics with npropa =    -1
 nv   =      4500 to the nearest 250
 nelt =      8500 to the nearest 250
 Mesh characteristics with npropa =     1
 nv   =      5000 to the nearest 250
 nelt =      9750 to the nearest 250
 Mesh characteristics with npropa =     5
 nv   =      2000 to the nearest 250
 nelt =      3750 to the nearest 250
```

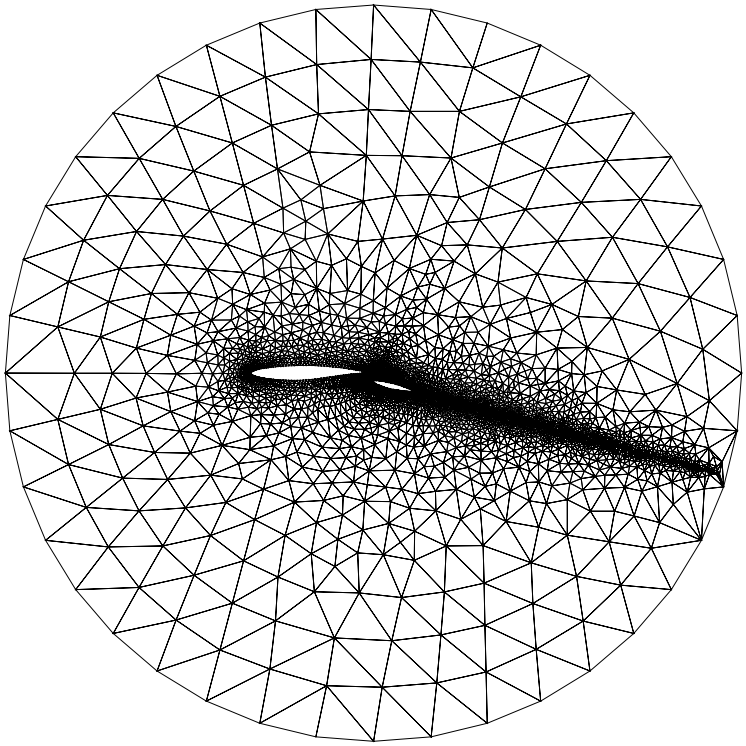
Example Program
Geometry for Generating Meshes



Mesh Generated Using Arithmetic Coefficient ω=1.2



Mesh Generated Using Arithmetic Coefficient ω=1.0



Mesh Generated Using Geometric Coefficient ω=1.0