NAG C Library Function Document

nag_mesh2d_front (d06acc)

1 Purpose

nag_mesh2d_front (d06acc) generates a triangular mesh of a closed polygonal region in $\mathbb{R}^2$, given a mesh of its boundary. It uses an Advancing Front process, based on an incremental method.

2 Specification

```
void nag_mesh2d_front (Integer nvb, Integer nvint, Integer nvmax, Integer nedge,
     const Integer edge[], Integer *nv, Integer *nelt, double coor[],
     Integer conn[], const double weight[], Integer itrace, const char *outfile,
     NagError *fail)
```

3 Description

nag_mesh2d_front (d06acc) generates the set of interior vertices using an Advancing Front process, based on an incremental method. It allows the user 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 Parameters

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 the 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: 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 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 edge and is not used by this routine. Note that the edge vertices are numbered from 1 to **nvb**.

Constraint: $1 \leq$ **EDGE**$(i, j) \leq$ **nvb** and **EDGE**$(1, j) \neq$ **EDGE**$(2, j)$ 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: 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 boundary mesh vertex, for $i = 1, \ldots,$ **nvb**. **COOR**$(1, i)$ contains the x-coordinate of the $(i -$ **nvb**$)$th fixed interior vertex, for $i =$ **nvb** $+ 1, \ldots,$ **nvb** + **nvint**. While **COOR**$(2, i)$ contains the corresponding y-coordinate, for $i = 1, \ldots,$ **nvb** + **nvint**.

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

9: **conn**[6 × **nvmax** + 15] – Integer *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 exit: 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**.

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: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_front (d06acc) as follows:

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

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

Users are advised to set **itrace** $= 0$, unless they are experienced with Finite Element meshes.

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

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

On entry, **nvint** $= \langle value \rangle$.
Constraint: **nvint** ≥ 0.

NE_INT_2

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

NE_INT_3

On entry, **nvb** + **nvint** > **nvmax**: **nv** $= \langle value \rangle$, **nvint** $= \langle value \rangle$, **nvmax** $= \langle value \rangle$.

NE_INT_4

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

NE_MESH_ERROR

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

NE_REAL_ARRAY_INPUT

On entry, **weight**$[i - 1] \leq 0.0$: **weight**$[i - 1] = \langle value \rangle$, $i = \langle value \rangle$.

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 $\langle value \rangle$.

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

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. 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)).

The user is 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**.

9 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 $(1.5, 0.0)$ with a radius 4.5, the first wing begins at the origin and it is normalised, finally the last wing is also normalised and begins at the point $(0.8, -0.3)$. 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 airfoils.

The boundary mesh has 120 vertices and 120 edges (see Figure 1 top). Note that the particular mesh generated could be sensitive to the machine precision and therefore may differ from one implementation to another. Contains the generated mesh Figure 1.

9.1 Program Text

```
/* nag_mesh2d_front (d06acc) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

/* Structure to allow data to be passed onto */
/* the d06bac user-supplied function fbnd */

struct user
{
  /* details of the double NACA0012 and the circle around it */

  double x0, y0, x1, y1, radius;
};

static double fbnd(Integer, double, double, Nag_Comm *);

#define EDGE(I,J) edge[3*((J)-1)+(I)-1]
#define LINED(I,J) lined[4*((J)-1)+(I)-1]
#define CONN(I,J) conn[3*((J)-1)+(I)-1]
#define COOR(I,J) coor[2*((J)-1)+(I)-1]
#define COORCH(I,J) coorch[2*((J)-1)+(I)-1]
#define COORUS(I,J) coorus[2*((J)-1)+(I)-1]

int main(void)
{
  const Integer nus=1, nvmax=2000, nedmx=200, nvint=40;
```

```
  struct user geom_Naca;
  double dnvint, radius, x0, x1, y0, y1;
  Integer exit_status, i, itrace, j, k, l, ncomp, nedge, nelt, nlines,
    nv, nvb, nvint2, reftk;
  char pmesh[2];
  double *coor=0, *coorch=0, *coorus=0, *rate=0,  *weight=0;
  Integer *conn=0, *edge=0, *lcomp=0, *lined=0, *nlcomp=0;
  NagError fail;
  Nag_Comm comm;

  INIT_FAIL(fail);
  exit_status = 0;

  Vprintf(" d06acc Example Program Results\n\n");

  /* Skip heading in data file */

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

  /* Initialise boundary mesh inputs: the number of lines and */
  /* the number of characteristic points of the boundary mesh */

  Vscanf("%ld", &nlines);
  Vscanf("%*[^\n] ");

  /* Allocate memory */

  if ( !(coor = NAG_ALLOC(2*nvmax, double)) ||
       !(coorch = NAG_ALLOC(2*nlines, double)) ||
       !(coorus = NAG_ALLOC(2*nus, double)) ||
       !(rate = NAG_ALLOC(nlines, double)) ||
       !(weight = NAG_ALLOC(nvint, double)) ||
       !(conn = NAG_ALLOC(3*(2*nvmax+5), Integer)) ||
       !(edge = NAG_ALLOC(3*nedmx, Integer)) ||
       !(lined = NAG_ALLOC(4*nlines, Integer)) ||
       !(lcomp = NAG_ALLOC(nlines, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* The double NACA0012 and the circle around it */

  for (j = 1; j <= nlines; ++j) Vscanf("%lf", &COORCH(1,j));
  Vscanf("%*[^\n] ");

  for (j = 1; j <= nlines; ++j) Vscanf("%lf", &COORCH(2,j));
  Vscanf("%*[^\n] ");

  /* The lines of the boundary mesh */

  for (j = 1; j <= nlines; ++j)
    {
      for (i = 1; i <= 4; ++i) Vscanf("%ld", &LINED(i,j));
      Vscanf("%lf", &rate[j-1]);
    }
  Vscanf("%*[^\n] ");

  /* The number of connected components to */
  /* the boundary and their information */

  Vscanf("%ld", &ncomp);
  Vscanf("%*[^\n] ");

  /* Allocate memory */

  if (!(nlcomp = NAG_ALLOC(ncomp, Integer)) )
    {
      Vprintf("Allocation failure\n");
      exit_status = -1;
```

```
      goto END;
    }

  j = 0;
  for (i = 0; i < ncomp; ++i)
    {
      Vscanf("%ld", &nlcomp[i]);
      Vscanf("%*[^\n] ");
      l = j + abs(nlcomp[i]);

      for (k = j; k < l; ++k) Vscanf("%ld", &lcomp[k]);
      Vscanf("%*[^\n] ");

      j += abs(nlcomp[i]);
    }

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

  /* Data passed to the user-supplied function */

  x0 = 1.5;
  y0 = 0.0;
  radius = 4.5;
  x1 = 0.8;
  y1 = -0.3;

  comm.p = (Pointer)

  geom_Naca.x0 = x0;
  geom_Naca.y0 = y0;
  geom_Naca.radius = radius;
  geom_Naca.x1 = x1;
  geom_Naca.y1 = y1;

  itrace = 0;

  /* Call to the 2D boundary mesh generator */

  d06bac(nlines, coorch, lined, fbnd, coorus, nus, rate, ncomp, nlcomp,
         lcomp, nvmax, nedmx, &nvb, coor, &nedge, edge, itrace, 0,
         &comm, &fail);

  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {

          Vprintf(" Boundary mesh characteristics\n");
          Vprintf(" nvb   =%6ld\n", nvb);
          Vprintf(" nedge =%6ld\n", nedge);
        }
      else if (pmesh[0] == 'Y')
        {

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

          Vprintf(" %10ld %10ld\n", nvb, nedge);

          for (i = 1; i <= nvb; ++i)
            Vprintf("  %4ld  %12.6e  %12.6e  \n",
                    i, COOR(1,i), COOR(2,i));

          for (i = 1; i <= nedge; ++i)
            Vprintf(" %4ld %4ld %4ld %4ld\n",
                    i, EDGE(1,i), EDGE(2,i), EDGE(3,i));
        }
      else
        {
          Vprintf("Problem with the printing option Y or N\n");
          exit_status = -1;
          goto END;
```

```
      }
    }
  else
    {
      Vprintf("Error from d06bac.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Initialise mesh control parameters */

  itrace = 0;

  /* Generation of interior vertices */
  /* for the wake of the first NACA */

  nvint2 = nvint/2;
  dnvint = 5.0/(nvint2 + 1.0);

  for (i = 1; i <= nvint2; ++i)
    {
      reftk = nvb + i;
      COOR(1, reftk) = i*dnvint + 1.0;
      COOR(2, reftk) = 0.0;
      weight[i-1] = 0.05;
    }

  /* for the wake of the second one */

  dnvint = 4.19/(nvint2 + 1.0);

  for (i = nvint2+1; i <= nvint; ++i)
    {
      reftk = nvb + i;
      COOR(1, reftk) = (i - nvint2)*dnvint + 1.8;
      COOR(2, reftk) = -0.3;
      weight[i-1] = 0.05;
    }

  /* Call to the 2D Advancing front mesh generator */

  d06acc(nvb, nvint, nvmax, nedge, edge, &nv, &nelt,
         coor, conn, weight, itrace, 0, &fail);

  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {
          Vprintf(" Complete mesh characteristics\n");
          Vprintf(" nv   =%6ld\n", nv);
          Vprintf(" nelt =%6ld\n", nelt);
        }
      else if (pmesh[0] == 'Y')
        {
          /* Output the 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 (k = 1; k <= nelt; ++k)
            Vprintf(" %10ld%10ld%10ld%10ld\n",
                    CONN(1,k), CONN(2,k), CONN(3,k), reftk);
        }
      else
        {
          Vprintf("Problem with the printing option Y or N\n");
          exit_status = -1;
          goto END;
```

```
        }
    }
  else
    {
      Vprintf("Error from d06acc.\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  if (coor) NAG_FREE(coor);
  if (coorch) NAG_FREE(coorch);
  if (coorus) NAG_FREE(coorus);
  if (rate) NAG_FREE(rate);
  if (weight) NAG_FREE(weight);
  if (conn) NAG_FREE(conn);
  if (edge) NAG_FREE(edge);
  if (lcomp) NAG_FREE(lcomp);
  if (lined) NAG_FREE(lined);
  if (nlcomp) NAG_FREE(nlcomp);

  return exit_status;
}

static double fbnd(Integer i, double x, double y, Nag_Comm *pcomm)
{
  double ret_val;
  double c, radius, x0, x1, y0, y1;
  struct user *geom_Naca = (struct user *)pcomm->p;

  x0 = geom_Naca->x0;
  y0 = geom_Naca->y0;
  radius = geom_Naca->radius;
  x1 = geom_Naca->x1;
  y1 = geom_Naca->y1;

  ret_val = 0.0;

  switch (i)
    {
    case 1:

      /* upper NACA0012 wing beginning at the origin */

      c = 1.008930411365;

      ret_val = 0.6*(0.2969*sqrt(c*x) - 0.126*(c*x) - 0.3516*pow(c*x,2.0)
                  + 0.2843*pow(c*x,3.0) - 0.1015*pow(c*x,4.0)) - c*y;
      break;

    case 2:

      /* lower NACA0012 wing beginning at the origin */

      c = 1.008930411365;

      ret_val = 0.6*(0.2969*sqrt(c*x) - 0.126*(c*x) - 0.3516*pow(c*x,2.0)
                  + 0.2843*pow(c*x,3.0) - 0.1015*pow(c*x,4.0)) + c*y;

      break;

    case 3:

      /* the circle around the double NACA */

      ret_val = (x-x0)*(x-x0) + (y-y0)*(y-y0) - radius*radius;
      break;

    case 4:

      /* upper NACA0012 wing beginning at (X1;Y1) */
```

```
      c = 1.008930411365;

      ret_val = 0.6*(0.2969*sqrt(c*(x-x1)) - 0.126*c*(x-x1) -
                     0.3516*pow(c*(x-x1),2.0) + 0.2843*pow(c*(x-x1),3.0) -
                     0.1015*pow(c*(x-x1),4.0)) - c*(y-y1);
      break;

    case 5:

      /* lower NACA0012 wing beginning at (X1;Y1) */

      c = 1.008930411365;

      ret_val = 0.6*(0.2969*sqrt(c*(x-x1)) - 0.126*(c*(x-x1)) -
                     0.3516*pow(c*(x-x1),2.0) + 0.2843*pow(c*(x-x1),3.0) -
                     0.1015*pow(c*(x-x1),4.0)) + c*(y-y1);
      break;
    }

  return ret_val;
}
```

9.2 Program Data

```
d06acc Example Program Data
 8                                                        :NLINES (m)
  0.0000   1.0000  -3.0000   6.0000   0.8000
  1.8000   1.5000   1.5000                          :(COORCH(1,1:m))
  0.0000   0.0000   0.0000   0.0000  -0.3000
 -0.3000   4.5000  -4.5000                          :(COORCH(2,1:m))
21  2  1  1   1.0000 21  1  2  2   1.0000
11  3  8  3   1.0000 11  4  7  3   1.0000
21  6  5  4   1.0000 21  5  6  5   1.0000
11  7  3  3   1.0000 11  8  4  3   1.0000 :(LINE(:,j),RATE(j),j=1,m)
 3                                     :NCOMP (n, number of contours)
-2                                      :number of lines in contour 1
 1  2                                              :lines of contour 1
 4                                      :number of lines in contour 2
 3  8  4  7                                        :lines of contour 2
-2                                      :number of lines in contour 3
 5  6                                              :lines of contour 3
'N'                                       :Printing option 'Y' or 'N'
```

9.3 Program Results

```
 d06acc Example Program Results

 Boundary mesh characteristics
 nvb   =   120
 nedge =   120
 Complete mesh characteristics
 nv    =  1892
 nelt  =  3666
```

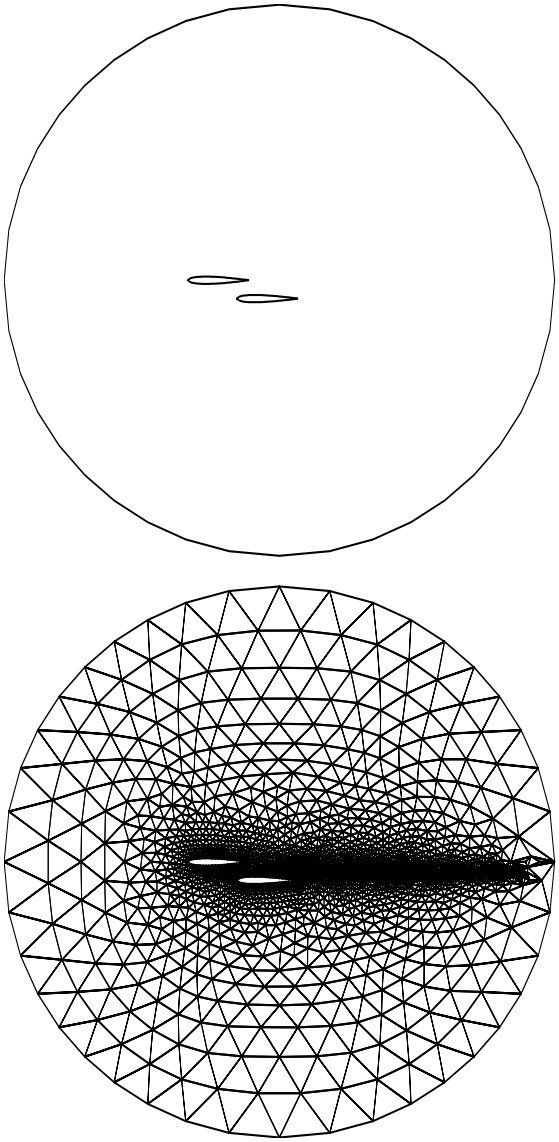

Figure 1
The boundary mesh (top), the interior mesh (bottom) of a double wing inside a circle geometry