NAG C Library Function Document

nag_mesh2d_bound (d06bac)

1 Purpose

nag_mesh2d_bound (d06bac) generates a boundary mesh on a closed connected subdomain Ω of $\mathbb{R}^2$.

2 Specification

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

void nag_mesh2d_bound (Integer nlines, const double coorch[], const Integer lined[],
      double (*fbnd)(Integer i, double x, double y, Nag_Comm *comm),
      const double coorus[], Integer nus, const double rate[], Integer ncomp,
      const Integer nlcomp[], const Integer lcomp[], Integer nvmax,
      Integer nedmx, Integer *nvb, double coor[], Integer *nedge, Integer edge[],
      Integer itrace, const char *outfile, Nag_Comm *comm, NagError *fail)
```

3 Description

Given a closed connected subdomain Ω of $\mathbb{R}^2$, whose boundary $\partial\Omega$ is divided by characteristic points into m distinct line segments, nag_mesh2d_bound (d06bac) generates a boundary mesh on $\partial\Omega$. Each line segment may be a straight line, a curve defined by the equation $f(x,y) = 0$, or a polygonal curve defined by a set of given boundary mesh points.

This function is primarily designed for use with either nag_mesh2d_inc (d06aac) (a simple incremental method) or nag_mesh2d_delaunay (d06abc) (Delaunay–Voronoi method) or nag_mesh2d_front (d06acc) (Advancing Front method) to triangulate the interior of the domain Ω. For more details about the boundary and interior mesh generation, 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: **nlines** – Integer *Input*

On entry: m, the number of lines that define the boundary of the closed connected subdomain (this equals the number of characteristic points which separate the entire boundary $\partial\Omega$ into lines).

Constraint: **nlines** ≥ 1.

2: **coorch**[**2** $\times$ **nlines**] – const double *Input*

On entry: **coorch**$[2 \times (i-1)]$ contains the x co-ordinate of the ith characteristic point, for $i = 1, \ldots,$ **nlines**; while **coorch**$[2 \times (i-1) + 1]$ contains the corresponding y co-ordinate.

3: **lined**[**4** $\times$ **nlines**] – const Integer *Input*

On entry: the description of the lines that define the boundary domain. The line i, for $i = 1, \ldots, m$, is defined as follows:

lined$[4 \times (i-1)]$

The number of points on the line, including two end points.

lined$[4 \times (i-1)+1]$

The first end point of the line. If **lined**$[4 \times (i-1)+1] = j$, then the co-ordinates of the first end point are those stored in **coorch**$[2 \times (j-1) : 2 \times (j-1)+1]$.

lined$[4 \times (i-1)+2]$

The second end point of the line. If **lined**$[4 \times (i-1)+2] = k$, then the co-ordinates of the second end point are those stored in **coorch**$[2 \times (k-1) : 2 \times (k-1)+1]$.

lined$[4 \times (i-1)+3]$

This defines the type of line segment connecting the end points. Additional information is conveyed by the numerical value of **lined**$[4 \times (i-1)+3]$ as follows:

(i) **lined**$[4 \times (i-1)+3] > 0$, the line is described in the user-supplied function **fbnd** with **lined**$[4 \times (i-1)+3]$ as the index. In this case, the line must be described in the trigonometric (anticlockwise) direction;

(ii) **lined**$[4 \times (i-1)+3] = 0$, the line is a straight line;

(iii) if **lined**$[4 \times (i-1)+3] < 0$, say (i.e., **lined**$[4 \times (i-1)+3] = -p$ for some index p), then the line is a polygonal arc joining the end points and interior points specified in **coorus**. In this case the line contains the points whose co-ordinates are stored in **coorch**$[2 \times (j-1) : 2 \times (j-1)+1]$,**coorus**$[2 \times (p-1) : 2 \times (p-1)+1]$, **coorus**$[2 \times p : 2 \times p+1]$,...,**coorus**$[2 \times (p+r-4) : 2 \times (p+r-4)+1]$, **coorch**$[2 \times (k-1) : 2 \times (k-1)+1]$, where $r =$ **lined**$[4 \times (i-1)]$, $j =$ **lined**$[4 \times (i-1)+1]$ and $k =$ **lined**$[4 \times (i-1)+2]$.

Constraints:

$2 \leq$ **lined**$[4 \times (i-1)]$;
$1 \leq$ **lined**$[4 \times (i-1)+1] \leq$ **nlines**;
$1 \leq$ **lined**$[4 \times (i-1)+2] \leq$ **nlines**;
lined$[4 \times (i-1)+1] \neq$ **lined**$[4 \times (i-1)+2]$, for $i = 1, 2, \ldots,$ **nlines**.

For each line described by the user-supplied function (lines with **lined**$[4 \times (i-1)+3] > 0$, $i = 1, \ldots,$ **nlines**) the two end points (**lined**$[4 \times (i-1)+1]$ and **lined**$[4 \times (i-1)+2]$) lie on the curve defined by index **lined**$[4 \times (i-1)+3]$ in the user-supplied function **fbnd**, i.e.,

$$\mathbf{fbnd}(\mathbf{lined}[4 \times (i-1)+3], \mathbf{coorch}[2 \times (\mathbf{lined}[4 \times (i-1)+1]-1)], \mathbf{coorch}[2 \times (\mathbf{lined}[4 \times (i-1)+1]-1)+1], \mathbf{comm}) = 0;$$
$$\mathbf{fbnd}(\mathbf{lined}[4 \times (i-1)+3], \mathbf{coorch}[2 \times (\mathbf{lined}[4 \times (i-1)+2]-1)], \mathbf{coorch}[2 \times (\mathbf{lined}[4 \times (i-1)+2]-1)+1], \mathbf{comm}) = 0, \text{ for } i = 1, 2, \ldots, \mathbf{nlines}.$$

For all lines described as polygonal arcs (lines with **lined**$[4 \times (i-1)+3] < 0$, $i = 1, \ldots,$ **nlines**) the sets of intermediate points (i.e., $[-\mathbf{lined}[4 \times (i-1)+3] : -\mathbf{lined}[4 \times (i-1)+3] + \mathbf{lined}[4 \times (i-1)] - 3]$ for all i such that **lined**$[4 \times (i-1)+3] < 0$) are not overlapping. This can be expressed as:

$$-\mathbf{lined}[4 \times (i-1)+3] + \mathbf{lined}[4 \times (i-1)] - 3 = \sum_{\{i, \mathbf{lined}[4 \times (i-1)+3] < 0\}} \{\mathbf{lined}[4 \times (i-1)] - 2\}$$

or

$$-\mathbf{lined}[4 \times (i-1)+3] + \mathbf{lined}[4 \times (i-1)] - 2 = -\mathbf{lined}[4 \times (j-1)+3],$$

for a j such that $j = 1, \ldots,$ **nlines**, $j \neq i$ and **lined**$[4 \times (j-1)+3] < 0$

4: **fbnd** – function, supplied by the user *External Function*

fbnd must be supplied by you to calculate the value of the function which describes the curve $\{(x,y) \in \mathbb{R}^2; \text{ such that } f(x,y) = 0\}$ on segments of the boundary for which

lined$[4 \times (i-1)+3] > 0$. If there are no boundaries for which **lined**$[4 \times (i-1)+3] > 0$ **fbnd** will never be referenced by nag_mesh2d_bound (d06bac) and **fbnd** may be null.

Its specification is:

```
double fbnd (Integer i, double x, double y, Nag_Comm *comm)
```

1: **i** – Integer *Input*

On entry: **lined**$[4 \times (i-1)+3]$, the reference index of the line (portion of the contour) i described.

2: **x** – double *Input*
3: **y** – double *Input*

On entry: the values of x and y at which $f(x,y)$ is to be evaluated.

4: **comm** – Nag_Comm * *Communication Structure*

Pointer to structure of type **Nag_Comm**; the following members are relevant to **fbnd**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_mesh2d_bound (d06bac) these pointers may be allocated memory by the user and initialized with various quantities for use by **fbnd** when called from nag_mesh2d_bound (d06bac).

5: **coorus**$[\mathbf{2} \times \mathbf{nus}]$ – const double *Input*

On entry: the co-ordinates of the intermediate points for polygonal arc lines. For a line i defined as a polygonal arc (i.e., **lined**$[4 \times (i-1)+3] < 0$), if $p = -$**lined**$[4 \times (i-1)+3]$, then **coorus**$[2 \times (k-1)]$, $k = p, p+1, \ldots, p+$**lined**$[4 \times (i-1)]-3$ must contain the x co-ordinate of the consecutive intermediate points for this line. Similarly **coorus**$[2 \times (k-1)+1]$, $k = p, p+1, \ldots, p+$**lined**$[4 \times (i-1)]-3$ must contain the corresponding y co-ordinate.

6: **nus** – Integer *Input*

On entry: the second dimension of the array **coorus** as declared in the function from which nag_mesh2d_bound (d06bac) is called.

Constraint: $\mathbf{nus} \geq \sum_{\{i, \mathbf{lined}[4\times(i-1)+3]<0\}} \{\mathbf{lined}[4 \times (i-1)] - 2\}$.

7: **rate**$[\mathbf{nlines}]$ – const double *Input*

On entry: **rate**$[i-1]$ is the geometric progression ratio between the points to be generated on the line i, for $i = 1, \ldots, m$ and **lined**$[4 \times (i-1)+3] \geq 0$. **rate**$[i-1]$ is not referenced if **lined**$[4 \times (i-1)+3] < 0$.

Constraint: if **lined**$[4 \times (i-1)+3] \geq 0$, **rate**$[i-1] > 0$, for $i = 1, 2, \ldots,$ **nlines**.

8: **ncomp** – Integer *Input*

On entry: n, the number of separately connected components of the boundary.

Constraint: **ncomp** ≥ 1.

9: **nlcomp**$[\mathbf{ncomp}]$ – const Integer *Input*

On entry: $|\mathbf{nlcomp}[k-1]|$ is the number of line segments in component k of the contour. The line i of component k runs in the direction **lined**$[4 \times (i-1)+1]$ to **lined**$[4 \times (i-1)+2]$ if **nlcomp**$[k-1] > 0$, and in the opposite direction otherwise; for $k = 1, \ldots, n$.

Constraints:

$1 \leq |\mathbf{nlcomp}[k-1]| \leq \mathbf{nlines}$, for $k = 1, 2, \ldots, \mathbf{ncomp}$;

$$\sum_{k=1}^{n} |\mathbf{nlcomp}[k-1]| = \mathbf{nlines}.$$

10: **lcomp**[**nlines**] – const Integer *Input*

On entry: **lcomp**$[l1 : l2]$, where $l2 = \sum_{i=1}^{k} |\mathbf{nlcomp}[i-1]|$ and $l1 = l2 + 1 - |\mathbf{nlcomp}[k-1]|$ is the list of line numbers for the kth components of the boundary, for $k = 1, \ldots, \mathbf{ncomp}$.

Constraint: **lcomp** must hold a valid permutation of the integers $[1, \mathbf{nlines}]$.

11: **nvmax** – Integer *Input*

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

Constraint: $\mathbf{nvmax} \geq \mathbf{nlines}$.

12: **nedmx** – Integer *Input*

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

Constraint: $\mathbf{nedmx} \geq 1$.

13: **nvb** – Integer * *Output*

On exit: the total number of boundary mesh vertices generated.

14: **coor**[**2** × **nvmax**] – double *Output*

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

15: **nedge** – Integer * *Output*

On exit: the total number of boundary edges in the boundary mesh.

16: **edge**[**3** × **nedmx**] – Integer *Output*

On exit: the specification of the boundary edges. $\mathbf{edge}[3 \times (j-1)]$ and $\mathbf{edge}[3 \times (j-1) + 1]$ will contain the vertex numbers of the two end points of the jth boundary edge. $\mathbf{edge}[3 \times (j-1) + 2]$ is a reference number for the jth boundary edge and

$\mathbf{edge}[3 \times (j-1) + 2] = \mathbf{lined}[4 \times (i-1) + 3]$, where i and j are such that the jth edges is part of the ith line of the boundary and $\mathbf{lined}[4 \times (i-1) + 3] \geq 0$;

$\mathbf{edge}[3 \times (j-1) + 2] = 100 + |\mathbf{lined}[4 \times (i-1) + 3]|$, where i and j are such that the jth edges is part of the ith line of the boundary and $\mathbf{lined}[4 \times (i-1) + 3] < 0$.

Note that the edge vertices are numbered from 1 to **nvb**.

17: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_bound (d06bac).

$\mathbf{itrace} \leq 0$

No output is generated.

$\mathbf{itrace} = 1$

Output from the boundary mesh generator is printed. This output contains the input information of each line and each connected component of the boundary.

itrace > 1

The output is similar to that produced when **itrace** $= 1$, but the co-ordinates of the generated vertices on the boundary are also output.

itrace $= -1$

An analysis of the output boundary mesh is printed on the current advisory message unit. This analysis includes the orientation (clockwise or anticlockwise) of each connected component of the boundary. Those informations could be of interest to you, especially if an interior meshing is carried out using the output of this function, calling either nag_mesh2d_inc (d06aac), nag_mesh2d_delaunay (d06abc) or nag_mesh2d_front (d06acc).

You are advised to set **itrace** $= 0$, unless you are experienced with Finite Element meshes generation.

18: **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.

19: **comm** – Nag_Comm * *Communication Structure*

The NAG communication argument (see Section 2.2.1.1 of the Essential Introduction).

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

The NAG error argument (see Section 2.6 of the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

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

NE_INT

On entry, **ncomp** the number of connected components of the boundary is less than 1: **ncomp** = ⟨*value*⟩.

On entry, **nedmx** the maximum number of boundary edge lines is less than 1: **nedmx** = ⟨*value*⟩.

On entry, **nlines** = ⟨*value*⟩.
Constraint: **nlines** ≥ 1.

NE_INT_2

On entry, **nus** $< nusmin$: **nus** = ⟨*value*⟩, $nusmin$ = ⟨*value*⟩.

On entry, **nvmax** the maximum number of boundary vertices is less than **nlines**: **nvmax** = ⟨*value*⟩, **nlines** = ⟨*value*⟩.

On entry, the line list for the separate connected component of the boundary is badly set: **lcomp**$[l-1]$ = ⟨*value*⟩, l = ⟨*value*⟩. It should be less than or equal to **nlines** and greater than or equal to 1.

On entry, the number of points on line ⟨*value*⟩ is ⟨*value*⟩. It should be greater than or equal to 2.

On entry, there is a correlation problem between the user-supplied co-ordinates and the specification of the polygonal arc representing line i = ⟨*value*⟩ with the index in **coorus** = ⟨*value*⟩.

On entry, the sum of absolute values of all numbers of line segments is different from **nlines**. The sum of all the elements of **nlcomp** = ⟨*value*⟩. **nlines** = ⟨*value*⟩.

NE_INT_3

On entry, the absolute number of line segments in the kth component of the contour should be less than or equal to **nlines** and greater than 0. $k = \langle value \rangle$, **nlcomp**$[k-1] = \langle value \rangle$, **nlines** $= \langle value \rangle$.

On entry, the index of the first end point of line $\langle value \rangle$ is $\langle value \rangle$. It should be greater than or equal to 1 and less than or equal to **nlines** $= \langle value \rangle$.

On entry, the index of the second end point of line $\langle value \rangle$ is $\langle value \rangle$. It should be greater than or equal to 1 and less than or equal to **nlines** $= \langle value \rangle$.

On entry, the indices of the extremities of line $\langle value \rangle$ are both equal to $\langle value \rangle$.

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.

NE_MESH_ERROR

An error has occurred during the generation of the boundary mesh. It appears that **nedmx** is not large enough: **nedmx** $= \langle value \rangle$.

An error has occurred during the generation of the boundary mesh. It appears that **nvmax** is not large enough: **nvmax** $= \langle value \rangle$.

On entry, end point 1, with index k, does not lie on the curve representing line i with index j: $k = \langle value \rangle$, $i = \langle value \rangle$, $j = \langle value \rangle$, $f(x, y) = \langle value \rangle$.

On entry, end point 2, with index k, does not lie on the curve representing line i with index j: $k = \langle value \rangle$, $i = \langle value \rangle$, $j = \langle value \rangle$, $f(x, y) = \langle value \rangle$.

On entry, the geometric progression ratio between the points to be generated on line $\langle value \rangle$ is $\langle value \rangle$. It should be greater than 0 unless the line segment is defined by user-supplied points.

On entry, there is a problem with either the co-ordinates of characteristic points, or with the definition of the mesh lines.

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 Further Comments

The boundary mesh generation technique in this function has a 'tree' structure. The boundary should be partitioned into geometrically simple segments (straight lines or curves) delimited by characteristic points. Then, the lines should be assembled into connected component of the boundary domain.

Using this strategy, the inputs to that function can be built up, following the requirements stated in Section 5:

the characteristic and the user-supplied intermediate points:

nlines, **nus**, **coorch** and **coorus**;

the characteristic lines:

lined, **fbnd**, **rate**;

finally the assembly of lines into the connected components of the boundary:

ncomp, and

nlcomp, **lcomp**.

The example below details the use of this strategy.

9 Example

The NAG logo is taken as an example of a geometry with holes. The boundary has been partitioned in 40 lines characteristic points; including 4 for the exterior boundary and 36 for the logo itself. All line geometry specifications have been considered, see the argument description of **lined**, including 4 lines defined as polygonal arc, 4 defined by a user-supplied function and all the others are straight lines.

Figure 1 top represents the boundary mesh of the NAG logo; there are 259 nodes and 259 edges. The Figure 1 middle and bottom represent the final mesh built using respectively the Delaunay–Voronoi (nag_mesh2d_delaunay (d06abc)) and the Advancing front (nag_mesh2d_front (d06acc)) method.

9.1 Program Text

```
/* nag_mesh2d_bound (d06bac) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 */

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

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

struct user
{
  /* details of the ellipse containing the NAG logo */

  double xa, xb, x0, y0;
};
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(int argc, char* argv[])
{
  const Integer nus=4, nvmax=1000, nedmx=300, nvint=0;
  struct user ellipse;
  Nag_Comm comm;
  double x0, xa, xb, xmax, xmin, y0, ymax, ymin;
  Integer exit_status, i, itrace, j, k, ncomp, nedge, nelt, nlines,
    npropa, nv, nvb, reftk, l;
  NagError fail;
  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;

  INIT_FAIL(fail);
  exit_status = 0;

  Vprintf(" nag_mesh2d_bound (d06bac) Example Program Results\n\n");
```

```
/* Skip heading in data file */

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

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

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

/* 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(1, 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 ellipse boundary which envelops  */
/* the NAG Logo, the N, the A and the G */

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] ");

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

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

/* 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 */

xmin = COORCH(1, 4);
xmax = COORCH(1, 2);
ymin = COORCH(2, 1);
ymax = COORCH(2, 3);

xa = (xmax-xmin)/2.0;
xb = (ymax-ymin)/2.0;

x0 = (xmin+xmax)/2.0;
y0 = (ymin+ymax)/2.0;

comm.p = (Pointer)

ellipse.xa = xa;
ellipse.xb = xb;
ellipse.x0 = x0;
ellipse.y0 = y0;

itrace = -1;

/* Call to the boundary mesh generator */

/* nag_mesh2d_bound (d06bac).
 * Generates a boundary mesh
 */
nag_mesh2d_bound(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 nag_mesh2d_bound (d06bac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

/* Initialise mesh control parameters */

itrace = 0;
npropa = 1;

/* 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')
      {
        Vprintf(" Complete mesh characteristics (Delaunay-Voronoi)\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 nag_mesh2d_delaunay (d06abc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

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

/* nag_mesh2d_front (d06acc).
 * Generates a two-dimensional mesh using an Advancing-front
 * method
 */
nag_mesh2d_front(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 (Advancing Front)\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 nag_mesh2d_front (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, d1, d2;
  double radius2, x0, xa, xb, y0;
  struct user *ellipse = (struct user *)pcomm->p;

  xa = ellipse->xa;
  xb = ellipse->xb;
  x0 = ellipse->x0;
  y0 = ellipse->y0;

  ret_val = 0.0;

  switch(i)
    {
    case 1:

      /* line 1,2,3, and 4: ellipse centred in (X0,Y0) with */
      /* XA and XB as coefficients */

      d1 = (x - x0)/xa;
      d2 = (y - y0)/xb;

      ret_val = d1*d1 + d2*d2 - 1.0;
      break;
```

```
    case 2:

      /* line 24, 27, 33 and 38 are a circle centred in (X0,Y0) */
      /* with radius SQRT(RADIUS2) */

      x0 = 20.5;
      y0 = 4.0;
      radius2 = 4.25;

      d1 = x - x0;
      d2 = y - y0;

      ret_val = d1*d1 + d2*d2 - radius2;
      break;

    case 3:

      x0 = 17.0;
      y0 = 8.5;
      radius2 = 5.0;

      d1 = x - x0;
      d2 = y - y0;

      ret_val = d1*d1 + d2*d2 - radius2;
      break;

    case 4:

      x0 = 17.0;
      y0 = 8.5;
      radius2 = 5.0;

      d1 = x - x0;
      d2 = y - y0;

      ret_val = d1*d1 + d2*d2 - radius2;
      break;

    case 5:

      x0 = 19.5;
      y0 = 4.0;
      radius2 = 1.25;

      d1 = x - x0;
      d2 = y - y0;

      ret_val = d1*d1 + d2*d2 - radius2;
      break;

    default:
      break;
    }

  return ret_val;
}
```

9.2 Program Data

```
nag_mesh2d_bound (d06bac) Example Program Data
40                                                        :NLINES (m)
  9.5000  33.0000   9.5000 -14.0000  -4.0000  -2.0000   2.0000
  4.0000   2.0000  -2.0000  -4.0000  -2.0000   2.0000   4.0000
  7.0000   9.0000  13.0000  16.0000   9.0000  12.0000   7.0000
 10.0000  18.0000  21.0000  17.0000  20.0000  16.0000  20.0000
 15.5000  16.0000  18.0000  21.0000  16.0000  18.0000  18.5811
 21.0000  17.0000  20.0000  20.5000  23.0000           :(COORCH(1,1:m))
 -1.0000   7.5000  16.0000   7.5000   3.0000   3.0000   3.0000
```

```
  3.0000   7.0000   8.0000  12.0000  12.0000  12.0000  12.0000
  3.0000   3.0000   3.0000   3.0000   5.0000   5.0000  12.0000
 12.0000   2.0000   2.0000   3.0000   3.0000   5.0000   5.0000
  6.0000   6.0000   6.0000   6.0000   6.5000   6.5000  10.0811
 10.0811  10.7361  10.7361  12.0000  12.0000          :(COORCH(2,1:m))
 -2.6667  -3.3333   3.3333   2.6667                   :(COORUS(1,1:4))
  3.0000   3.0000   3.0000   3.0000                   :(COORUS(2,1:4))
15  1  2  1   0.9500   15  2  3  1   1.0500
15  3  4  1   0.9500   15  4  1  1   1.0500
 4  6  5 -1   1.0000   10 10  6  0   1.0000
10  7 10  0   1.0000    4  8  7 -3   1.0000
15 14  8  0   1.0000    4 13 14  0   1.0000
10  9 13  0   1.0000   10 12  9  0   1.0000
 4 11 12  0   1.0000   15  5 11  0   1.0000
 4 16 15  0   1.0000    7 19 16  0   1.0000
 4 20 19  0   1.0000    7 17 20  0   1.0000
 4 18 17  0   1.0000   13 22 18  0   1.0000
 5 21 22  0   1.0000   13 15 21  0   1.0000
 4 24 23  0   1.0000   10 24 32  2   1.0000
 4 31 32  0   1.0000    4 34 31  0   1.0000
10 34 35  3   1.0000    4 36 35  0   1.0000
 4 40 36  0   1.0000    4 39 40  0   1.0000
 4 38 39  0   1.0000    4 37 38  0   1.0000
10 37 33  4   1.0000    4 30 33  0   1.0000
 4 29 30  0   1.0000    4 27 29  0   1.0000
 4 28 27  0   1.0000   10 26 28  5   1.0000
 4 25 26  0   1.0000    4 23 25  0   1.0000 :(LINE(:,j),RATE(j),j=1,m)
 4                                        :NCOMP (n, number of contours)
 4                                         :number of lines in contour 1
 1  2  3  4                                          :lines of contour 1
10                                         :number of lines in contour 2
14 13 12 11 10  9  8  7  6  5                        :lines of contour 2
 8                                         :number of lines in contour 3
22 21 20 19 18 17 16 15                              :lines of contour 3
18                                         :number of lines in contour 4
30 29 28 27 26 25 24 23 40 39
38 37 36 35 34 33 32 31                              :lines of contour 4
'N'                                          :Printing option 'Y' or 'N'
```

9.3 Program Results

```
nag_mesh2d_bound (d06bac) Example Program Results

ANALYSIS OF THE BOUNDARY CREATED:
THE BOUNDARY MESH CONTAINS    259 VERTEX AND    259 EDGES
THERE ARE      4 COMPONENTS CONNECTED THE BOUNDARY
THE  1-st COMPONENT CONTAINS      4 LINES IN ANTICLOCKWISE ORIENTATION
THE  2-nd COMPONENT CONTAINS     10 LINES IN CLOCKWISE ORIENTATION
THE  3-rd COMPONENT CONTAINS      8 LINES IN CLOCKWISE ORIENTATION
THE  4-th COMPONENT CONTAINS     18 LINES IN CLOCKWISE ORIENTATION
Boundary mesh characteristics
nvb   =   259
nedge =   259
Complete mesh characteristics (Delaunay-Voronoi)
nv   =   652
nelt =  1049
Complete mesh characteristics (Advancing Front)
nv   =   662
nelt =  1069
```

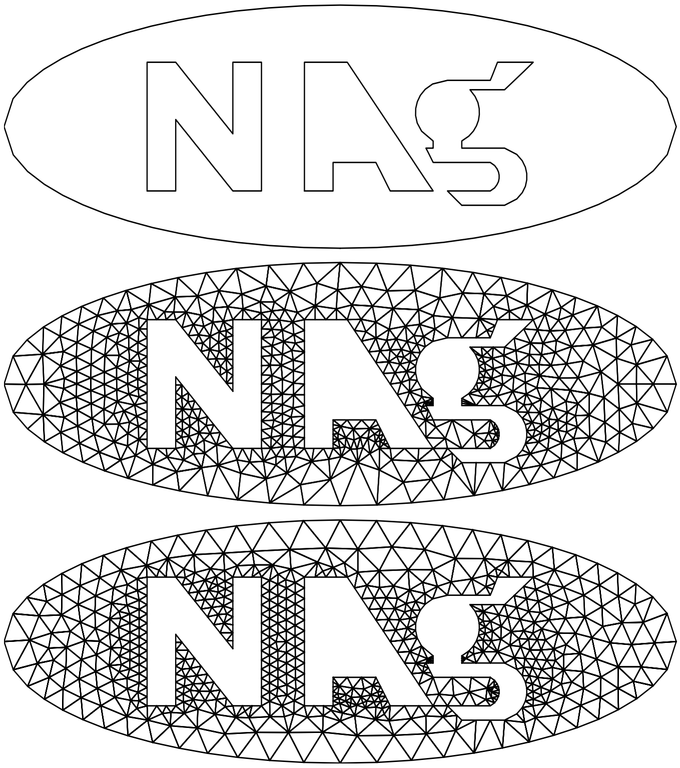

Figure 1
The boundary mesh (top), the Delaunay–Voronoi mesh (middle) and the Advancing Front mesh (bottom) of the NAG logo geometry