NAG 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

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

```
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 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: $\mathbf{nvb} \geq 3$.

2: **nvint** – Integer *Input*

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

Constraint: $\mathbf{nvint} \geq 0$.

3: **nvmax** – Integer *Input*

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

Constraint: $\mathbf{nvmax} \geq \mathbf{nvb} + \mathbf{nvint}$.

4: **nedge** – Integer *Input*

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

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

5: **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_front (d06acc). 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** $\times$ **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** $\times$ (**2** $\times$ **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: **itrace** – Integer *Input*

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

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.

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 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument $\langle value \rangle$ had an illegal value.

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: $J = \langle value \rangle$ and $I = \langle value \rangle$.

NE_INT_3

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

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

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 3.6.6 in the Essential Introduction 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 3.6.5 in the Essential Introduction 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_front (d06acc) is not threaded by NAG in any implementation.

nag_mesh2d_front (d06acc) 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. 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)).

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 $(1.5, 0.0)$ with a radius 4.5, the first wing begins at the origin and it is normalized, finally the last wing is also normalized 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.

10.1 Program Text

```
/* nag_mesh2d_front (d06acc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

#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 nag_mesh2d_bound (d06bac) user-supplied function fbnd */

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

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

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL fbnd(Integer, double, double, Nag_Comm *);
#ifdef __cplusplus
}
#endif

#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 = 0, i, itrace, j, k, l, ncomp, nedge, nelt,
                nlines;
  Integer       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);

  printf(" nag_mesh2d_front (d06acc) Example Program Results\n\n");

  /* Skip heading in data file */

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


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

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

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

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

#ifdef _WIN32
  for (j = 1; j <= nlines; ++j) scanf_s("%lf", &COORCH(1, j));
#else
  for (j = 1; j <= nlines; ++j) scanf("%lf", &COORCH(1, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  for (j = 1; j <= nlines; ++j) scanf_s("%lf", &COORCH(2, j));
#else
  for (j = 1; j <= nlines; ++j) scanf("%lf", &COORCH(2, j));
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* The lines of the boundary mesh */

  for (j = 1; j <= nlines; ++j)
    {
#ifdef _WIN32
      for (i = 1; i <= 4; ++i) scanf_s("%"NAG_IFMT"", &LINED(i, j));
#else
      for (i = 1; i <= 4; ++i) scanf("%"NAG_IFMT"", &LINED(i, j));
#endif
#ifdef _WIN32
      scanf_s("%lf", &rate[j-1]);
#else
      scanf("%lf", &rate[j-1]);
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
```

```
  scanf("%*[^\n] ");
#endif

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

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

  /* Allocate memory */

  if (!(nlcomp = NAG_ALLOC(ncomp, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  j = 0;
  for (i = 0; i < ncomp; ++i)
    {
#ifdef _WIN32
      scanf_s("%"NAG_IFMT"", &nlcomp[i]);
#else
      scanf("%"NAG_IFMT"", &nlcomp[i]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif
      l = j + abs(nlcomp[i]);

#ifdef _WIN32
      for (k = j; k < l; ++k) scanf_s("%"NAG_IFMT"", &lcomp[k]);
#else
      for (k = j; k < l; ++k) scanf("%"NAG_IFMT"", &lcomp[k]);
#endif
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

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

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

  /* 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;
```

```
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 */

/* 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')
      {
        printf(" Boundary mesh characteristics\n");
        printf(" nvb   =%6"NAG_IFMT"\n", nvb);
        printf(" nedge =%6"NAG_IFMT"\n", nedge);
      }
    else if (pmesh[0] == 'Y')
      {
        /* Output the mesh to view it using the NAG Graphics Library */

        printf(" %10"NAG_IFMT" %10"NAG_IFMT"\n", nvb, nedge);

        for (i = 1; i <= nvb; ++i)
          printf("  %4"NAG_IFMT"  %15.6e  %15.6e  \n",
                 i, COOR(1, i), COOR(2, i));

        for (i = 1; i <= nedge; ++i)
          printf(" %4"NAG_IFMT" %4"NAG_IFMT" %4"NAG_IFMT" %4"NAG_IFMT"\n",
                 i, EDGE(1, i), EDGE(2, i), EDGE(3, i));
      }
    else
      {
        printf("Problem with the printing option Y or N\n");
        exit_status = -1;
        goto END;
      }
  }
else
  {
    printf("Error from nag_mesh2d_bound (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 */

  /* 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')
        {
          printf(" Complete mesh characteristics\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_front (d06acc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

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

```
  return exit_status;
}

double NAG_CALL 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;
}
```

10.2 Program Data

```
nag_mesh2d_front (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'
```

10.3 Program Results

```
 nag_mesh2d_front (d06acc) Example Program Results

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

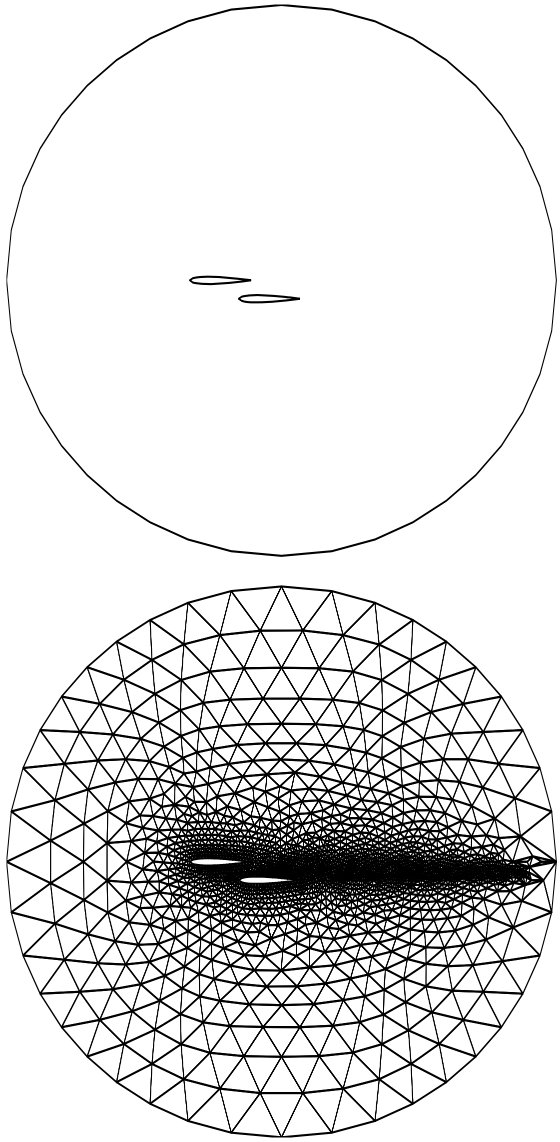

Figure 1
The boundary mesh (top), the interior mesh (bottom) of a double wing inside a circle geometry