NAG C Library Function Document

nag_mesh2d_smooth (d06cac)

1 Purpose

nag_mesh2d_smooth (d06cac) uses a barycentering technique to smooth a given mesh.

2 Specification

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

```
void nag_mesh2d_smooth (Integer nv, Integer nelt, Integer nedge, double coor[],
     const Integer edge[], const Integer conn[], Integer nvfix,
     const Integer numfix[], Integer itrace, const char *outfile, Integer nqint,
     NagError *fail)
```

3 Description

nag_mesh2d_smooth (d06cac) uses a barycentering approach to improve the smoothness of a given mesh. The measure of quality used for a triangle K is

$$Q_K = \alpha \frac{h_K}{\rho_K};$$

where h_K is the diameter (length of the longest edge) of K, ρ_K is the radius of its inscribed circle and $\alpha = \frac{\sqrt{3}}{6}$ is a normalization factor chosen to give $Q_K = 1$ for an equilateral triangle. Q_K ranges from 1, for an equilateral triangle, to ∞, for a totally flat triangle.

nag_mesh2d_smooth (d06cac) makes small perturbation to vertices (using a barycenter formula) in order to give a reasonably good value of Q_K for all neighbouring triangles. Some vertices may optionally be excluded from this process.

For more details about the smoothing method, especially with regard to differing quality, 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: **nv** – Integer *Input*

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

Constraint: $\mathbf{nv} \geq 3$.

2: **nelt** – Integer *Input*

On entry: the number of triangles in the input mesh.

Constraint: $\mathbf{nelt} \leq 2 \times \mathbf{nv} - 1$.

3: **nedge** – Integer *Input*

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

Constraint: **nedge** ≥ 1.

4: **coor**[**2** × **nv**] – double *Input/Output*

On entry: **coor**$[2 \times (i-1)]$ contains the x co-ordinate of the ith input mesh vertex, for $i = 1, \ldots,$ **nv**; while **coor**$[2 \times (i-1) + 1]$ contains the corresponding y co-ordinate.

On exit: **coor**$[2 \times (i-1)]$ will contain the x co-ordinate of the ith smoothed mesh vertex, for $i = 1, \ldots,$ **nv**; while **coor**$[2 \times (i-1) + 1]$ will contain the corresponding y co-ordinate. Note that the co-ordinates of boundary and interface edge vertices, as well as those specified by you (see the description of **numfix**), are unchanged by the process.

5: **edge**[**3** × **nedge**] – const Integer *Input*

On entry: the specification of the boundary or interface edges. **edge**$[3 \times (j-1)]$ and **edge**$[3 \times (j-1) + 1]$ contain the vertex numbers of the two end points of the jth boundary edge. **edge**$[3 \times (j-1) + 2]$ is a user-supplied tag for the jth boundary or interface edge: **edge**$[3 \times (j-1) + 2] = 0$ for an interior edge and has a non-zero tag otherwise. Note that the edge vertices are numbered from 1 to **nv**.

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

6: **conn**[**3** × **nelt**] – const Integer *Input*

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

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

7: **nvfix** – Integer *Input*

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

Constraint: $0 \leq$ **nvfix** $\leq$ **nv**.

8: **numfix**[dim] – const Integer *Input*

Note: the dimension, dim, of the array **numfix** must be at least max(1, **nvfix**).

On entry: the indices in **coor** of fixed interior vertices of the input mesh.

Constraint: if **nvfix** > 0, $1 \leq$ **numfix**$[i-1] \leq$ **nv**, for $i = 1, 2, \ldots,$ **nvfix**.

9: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_smooth (d06cac).

itrace ≤ 0

No output is generated.

itrace $= 1$

A histogram of the triangular element qualities is printed before and after smoothing. This histogram gives the lowest and the highest triangle quality as well as the number of elements lying in each of the **nqint** equal intervals between the extremes.

itrace > 1

The output is similar to that produced when **itrace** $= 1$ but the connectivity between vertices and triangles (for each vertex, the list of triangles in which it appears) is given.

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

10: **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.

11: **nqint** – Integer *Input*

On entry: the number of intervals between the extreme quality values for the input and the smoothed mesh. If **itrace** $= 0$, then **nqint** is not referenced.

12: **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 $\langle value \rangle$ had an illegal value.

NE_INT

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

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

NE_INT_2

On entry, **nelt** $= \langle value \rangle$, **nv** $= \langle value \rangle$.
Constraint: **nelt** $\leq 2 \times$ **nv** $- 1$.

On entry, **nv** $= \langle value \rangle$, **nvfix** $= \langle value \rangle$.
Constraint: $0 \leq$ **nvfix** $\leq$ **nv**.

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

On entry, vertices 1 and 2 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

On entry, vertices 1 and 3 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

On entry, vertices 2 and 3 of the triangle k have the same index i: $k = \langle value \rangle$, $i = \langle value \rangle$.

NE_INT_3

On entry, **numfix**$[i-1] < 1$ or **numfix**$[i-1] >$ **nv**: **numfix**$[i-1] = \langle value \rangle$, $i = \langle value \rangle$, **nv** $= \langle value \rangle$.

NE_INT_4

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

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

NE_INTERNAL_ERROR

A serious error has occurred in an internal call to an auxiliary function. Check the input mesh especially the connectivity. Seek expert help.

NE_NOT_CLOSE_FILE

Cannot close file ⟨*value*⟩.

NE_NOT_WRITE_FILE

Cannot open file ⟨*value*⟩ for writing.

7 Accuracy

Not applicable.

8 Further Comments

Not applicable.

9 Example

In this example, a uniform mesh on the unit square is randomly distorted using functions from Chapter g05 (see Figure 1). nag_mesh2d_smooth (d06cac) is then used to smooth the distorted mesh and recover a uniform mesh (see Figure 2).

9.1 Program Text

```
/* nag_mesh2d_smooth (d06cac) Example Program.
 *
 * Copyright 2001 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 *
 * Mark 8 revised, 2004
 *
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagd06.h>
#include <nagg05.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 nvfix=0;
  const double zero=0.0;
  double delta, hx, hy, pi, dpi, r, rad, sk, theta, x1, x2, x3, y1, y2, y3;
  Integer exit_status, i, imax, imaxm1, ind, itrace, j, jmax, jmaxm1, k,
    me1, me2, me3, nedge, nelt, nqint, nv, reftk;
  char pmesh[2];
  double *coor=0;
  Integer *conn=0, *edge=0, *numfix=0;
  /* nag_rngs_init_repeatable (g05kbc) requires iseed[0]= and igen=0 for
   * it to mimic nag_random_init_repeatable().
   */
  Integer igen = 0, iseed[] = {0, 0, 0, 0};
  double one_draw[1];
  NagError fail;
```

```
INIT_FAIL(fail);
exit_status = 0;

Vprintf(" nag_mesh2d_smooth (d06cac) Example Program Results\n\n");

/* Skip heading in data file */

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

/* Read imax and jmax, the number of vertices */
/* in the x and y directions respectively. */

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

/* Read distortion percentage and calculate radius */
/* of distortion neighbourhood so that cross-over */
/* can only occur at 100% or greater. */

Vscanf("%lf", &delta);
Vscanf("%*[^\n] ");

nv = imax*jmax;
imaxm1 = imax - 1;
jmaxm1 = jmax - 1;
nelt = 2*imaxm1*jmaxm1;
nedge = 2*(imaxm1 + jmaxm1);

/* Allocate memory */

if ( !(coor = NAG_ALLOC(2*nv, double)) ||
     !(conn = NAG_ALLOC(3*nelt, Integer)) ||
     !(edge = NAG_ALLOC(3*nedge, Integer)) ||
     !(numfix = NAG_ALLOC(1, Integer)) )
  {
    Vprintf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

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

hx = 1.0/(double)imaxm1;
hy = 1.0/(double)jmaxm1;
rad = 0.01*delta*(hx > hy ? hy : hx)/2.0;
pi = 4.0*atan(1.0);
/* nag_rngs_init_repeatable (g05kbc).
 * Initialize seeds of a given generator for random number
 * generating functions (that pass seeds explicitly) to give
 * a repeatable sequence
 */
nag_rngs_init_repeatable(&igen, iseed);

/* Generate a simple uniform mesh and then distort it */
/* randomly within the distortion neighbourhood of each */
/* node. */

ind= 0;
for (j = 1; j <= jmax; ++j)
  {
    for (i = 1; i <= imax; ++i)
      {
        /* nag_rngs_uniform (g05lgc).
         * Generates a vector of random numbers from a uniform
         * distribution, seeds and generator number passed
         * explicitly
         */
```

```
        nag_rngs_uniform(zero, rad, 1, one_draw, igen, iseed, NAGERR_DEFAULT);
        r = one_draw[0];

        dpi = 2.0*pi;
        /* nag_rngs_uniform (g05lgc), see above. */
        nag_rngs_uniform(zero, dpi, 1, one_draw, igen, iseed, NAGERR_DEFAULT);
        theta = one_draw[0];
        if (i == 1 || i == imax || j == 1 || j == jmax) r = 0.0;
        k = (j-1)*imax + i;
        COOR(1, k) = (i-1)*hx + r*cos(theta);
        COOR(2, k) = (j-1)*hy + r*sin(theta);
        if (i < imax && j < jmax)
          {
            ++ind;
            CONN(1, ind) = k;
            CONN(2, ind) = k + 1;
            CONN(3, ind) = k + imax + 1;
            ++ind;
            CONN(1, ind) = k;
            CONN(2, ind) = k + imax + 1;
            CONN(3, ind) = k + imax;
          }
      }
  }

if (pmesh[0] == 'N')
  {
    Vprintf(" The complete distorted 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));
  }
else
  {
    Vprintf("Problem with the printing option Y or N\n");
  }

reftk = 0;

for (k = 1; k <= nelt; ++k)
  {
    me1 = CONN(1, k);
    me2 = CONN(2, k);
    me3 = CONN(3, k);

    x1 = COOR(1, me1);
    x2 = COOR(1, me2);
    x3 = COOR(1, me3);
    y1 = COOR(2, me1);
    y2 = COOR(2, me2);
    y3 = COOR(2, me3);

    sk = 0.5*((x2-x1)*(y3-y1) - (y2-y1)*(x3-x1));
    if (sk < 0.0)
      {
        Vprintf("Error the surface of the element is negative\n");
        Vprintf(" k = %6ld\n", k);
        Vprintf(" sk = %12.6e\n", sk);
        exit_status = -1;
        goto END;
      }
```

```
      if (pmesh[0] == 'Y')
        Vprintf(" %10ld%10ld%10ld%10ld\n",
                CONN(1,k), CONN(2,k), CONN(3,k), reftk);
    }

  /* Boundary edges */

  ind = 0;
  for (i = 1; i <= imaxm1; ++i)
    {
      ++ind;
      EDGE(1,ind) = i;
      EDGE(2,ind) = i + 1;
      EDGE(3,ind) = 0;
    }

  for (i = 1; i <= jmaxm1; ++i)
    {
      ++ind;
      EDGE(1,ind) = i*imax;
      EDGE(2,ind) = (i+1)*imax;
      EDGE(3,ind) = 0;
    }

  for (i = 1; i <= (imax - 1); ++i)
    {
      ++ind;
      EDGE(1,ind) = imax*jmax - i + 1;
      EDGE(2,ind) = imax*jmax - i;
      EDGE(3,ind) = 0;
    }

  for (i = 1; i <= jmaxm1; ++i)
    {
      ++ind;
      EDGE(1,ind) = (jmax - i)*imax + 1;
      EDGE(2,ind) = (jmax - i - 1)*imax + 1;
      EDGE(3,ind) = 0;
    }

  itrace = 1;
  nqint = 10;

  /* Call the smoothing routine */

  /* nag_mesh2d_smooth (d06cac).
   * Uses a barycentering technique to smooth a given mesh
   */
  nag_mesh2d_smooth(nv, nelt, nedge, coor, edge, conn, nvfix, numfix, itrace,
                    0, nqint, &fail);

  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {
          Vprintf(" The complete smoothed 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 nag_mesh2d_smooth (d06cac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

 END:
  if (coor) NAG_FREE(coor);
  if (conn) NAG_FREE(conn);
  if (edge) NAG_FREE(edge);
  if (numfix) NAG_FREE(numfix);

  return exit_status;
}
```

9.2 Program Data

```
nag_mesh2d_smooth (d06cac) Example Program Data
20 20    :IMAX JMAX
87.0     :DELTA
'N'      :Printing option 'Y' or 'N'
```

9.3 Program Results

```
 nag_mesh2d_smooth (d06cac) Example Program Results

 The complete distorted mesh characteristics
 nv   =   400
 nelt =   722
 BEFORE SMOOTHING
 MINIMUM SMOOTHNESS MEASURE:       1.0048907
 MINIMUM SMOOTHNESS MEASURE:     133.2110681
 DISTRIBUTION
 INTERVAL                         NUMBER OF ELEMENTS
      1.0048907 -      14.2255084       720
     14.2255084 -      27.4461262         0
     27.4461262 -      40.6667439         0
     40.6667439 -      53.8873616         0
     53.8873616 -      67.1079794         0
     67.1079794 -      80.3285971         0
     80.3285971 -      93.5492149         0
     93.5492149 -     106.7698326         0
    106.7698326 -     119.9904504         0
    119.9904504 -     133.2110681         1

 AFTER SMOOTHING
 MINIMUM SMOOTHNESS MEASURE:       1.3346259
 MINIMUM SMOOTHNESS MEASURE:       1.4572261
 DISTRIBUTION
 INTERVAL                         NUMBER OF ELEMENTS
      1.3346259 -       1.3468859        10
      1.3468859 -       1.3591459        36
      1.3591459 -       1.3714060        46
      1.3714060 -       1.3836660       117
      1.3836660 -       1.3959260       186
      1.3959260 -       1.4081860       137
      1.4081860 -       1.4204460       106
      1.4204460 -       1.4327061        51
```

```
     1.4327061 -       1.4449661        28
     1.4449661 -       1.4572261         4
The complete smoothed mesh characteristics
nv   =   400
nelt =   722
```

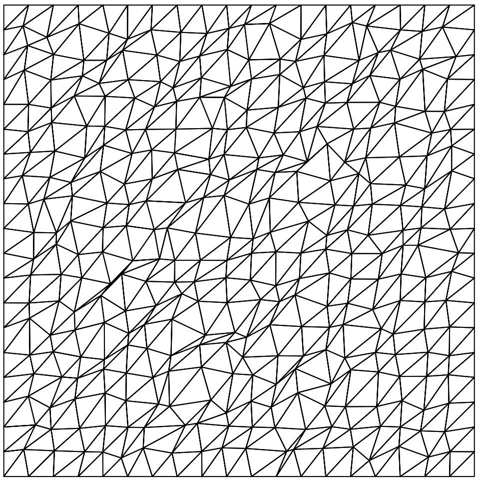

Figure 1
Distorted uniform mesh

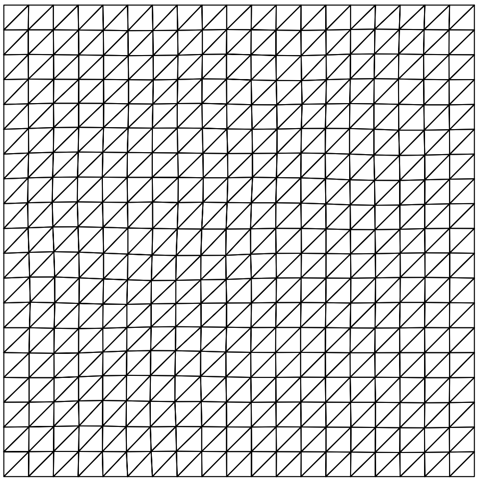

Figure 2
After smoothing with nag_mesh2d_smooth (d06cac)