NAG 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 $\alpha = \dfrac{\sqrt{3}}{6}$ 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: **nv** ≥ 3.

2: **nelt** – Integer *Input*

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

Constraint: **nelt** $\leq 2 \times$ **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** $\times$ **nv**] – 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 mesh vertex, for $i=1,2,\ldots,$ **nv**; while **coor**$[(i-1)\times 2+1]$ contains the corresponding y coordinate.

On exit: **coor**$[(i-1)\times 2]$ will contain the x coordinate of the ith smoothed mesh vertex, for $i=1,2,\ldots,$ **nv**; while **coor**$[(i-1)\times 2+1]$ will contain the corresponding y coordinate. Note that the coordinates 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** $\times$ **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 or interface 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 or interface edge: **edge**$[(j-1)\times 3+2]=0$ for an interior edge and has a nonzero tag otherwise. Note that the edge vertices are numbered from 1 to **nv**.

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

6: **conn**[**3** $\times$ **nelt**] – const Integer *Input*

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

On entry: 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**.

Constraint: $1\leq$ **conn**$[(j-1)\times 3+i-1]\leq$ **nv** and **conn**$[(j-1)\times 3]\neq$ **conn**$[(j-1)\times 3+1]$ and **conn**$[(j-1)\times 3]\neq$ **conn**$[(j-1)\times 3+2]$ and **conn**$[(j-1)\times 3+1]\neq$ **conn**$[(j-1)\times 3+2]$, 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** $= 0$, **nqint** is not referenced.

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

NE_INT_2

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

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

On entry, the endpoints of the edge J have the same index I: $J = \langle value \rangle$ and $I = \langle value \rangle$.

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

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

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

NE_INT_3

On entry, **numfix**$[I - 1] = \langle value \rangle$, $I = \langle value \rangle$ and **nv** $= \langle value \rangle$.
Constraint: **numfix**$[I - 1] \geq 1$ and **numfix**$[I - 1] \leq$ **nv**.

NE_INT_4

On entry, $\mathbf{CONN}(I,J) = \langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{nv} = \langle value \rangle$.
Constraint: $\mathbf{CONN}(I,J) \geq 1$ and $\mathbf{CONN}(I,J) \leq \mathbf{nv}$, where $\mathbf{CONN}(I,J)$ denotes $\mathbf{conn}[(J-1) \times 3 + I - 1]$.

On entry, $\mathbf{EDGE}(I,J) = \langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and $\mathbf{nv} = \langle value \rangle$.
Constraint: $\mathbf{EDGE}(I,J) \geq 1$ and $\mathbf{EDGE}(I,J) \leq \mathbf{nv}$, where $\mathbf{EDGE}(I,J)$ denotes $\mathbf{edge}[(J-1) \times 3 + I - 1]$.

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.

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

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.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_mesh2d_smooth (d06cac) is not threaded by NAG in any implementation.

nag_mesh2d_smooth (d06cac) 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

None.

10 Example

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

10.1 Program Text

```
/* nag_mesh2d_smooth (d06cac) Example Program.
 *
 * Copyright 2014 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)
{
  /* Integer scalar and array declarations */
  const Integer nvfix = 0;
  Integer       exit_status = 0;
  Integer       i, imax, imaxm1, ind, itrace, j, jmax, jmaxm1, k,
                me1, me2, me3, nedge, nelt, nqint, nv, reftk, lstate;
  Integer       *conn = 0, *edge = 0, *numfix = 0, *state = 0;

  /* Character scalar and array declarations */
  char          pmesh[2];

  /* NAG structures */
  NagError      fail;

  /* Double scalar and array declarations */
  double        delta, hx, hy, pi, dpi, r, rad, sk, theta, x1, x2, x3, y1, y2,
                y3;
  double        one_draw[1];
  double        *coor = 0;

  /* Choose the base generator */
  Nag_BaseRNG   genid = Nag_Basic;
  Integer       subid = 0;

  /* Set the seed */
  Integer       seed[] = { 1762543 };
  Integer       lseed = 1;

  /* Initialise the error structure */
  INIT_FAIL(fail);

  /* Get the length of the state array */
  lstate = -1;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  printf(" nag_mesh2d_smooth (d06cac) Example Program Results\n\n");
  fflush(stdout);

  /* Skip heading in data file */
#ifdef _WIN32
```

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

  /* Read imax and jmax, the number of vertices */
  /* in the x and y directions respectively. */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &imax);
#else
  scanf("%"NAG_IFMT"", &imax);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"", &jmax);
#else
  scanf("%"NAG_IFMT"", &jmax);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read distortion percentage and calculate radius */
  /* of distortion neighbourhood so that cross-over */
  /* can only occur at 100% or greater. */
#ifdef _WIN32
  scanf_s("%lf", &delta);
#else
  scanf("%lf", &delta);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  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)) ||
      !(state = NAG_ALLOC(lstate, Integer)) ||
      !(numfix = NAG_ALLOC(1, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

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

  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);
```

```
  /* Initialise the generator to a repeatable sequence */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

  /* 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)
        {
          /* Generate one uniform variate between 0 and rad */
          nag_rand_uniform(1, 0.0, rad, state, one_draw, &fail);
          if (fail.code != NE_NOERROR)
            {
              printf("Error from nag_rand_uniform (g05sqc).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }
          r = one_draw[0];

          dpi = 2.0*pi;

          /* Generate one uniform variate between 0 and dpi */
          nag_rand_uniform(1, 0.0, dpi, state, one_draw, &fail);
          if (fail.code != NE_NOERROR)
            {
              printf("Error from nag_rand_uniform (g05sqc).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }
          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')
    {
      printf(" The complete distorted mesh characteristics\n");
      printf(" nv   =%6"NAG_IFMT"\n", nv);
      printf(" nelt =%6"NAG_IFMT"\n\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));
  }
else
  {
    printf("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)
      {
        printf("Error the surface of the element is negative\n");
        printf(" k = %6"NAG_IFMT"\n", k);
        printf(" sk = %15.6e\n", sk);
        exit_status = -1;
        goto END;
      }

    if (pmesh[0] == 'Y')
      printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\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
   */
  fflush(stdout);
  nag_mesh2d_smooth(nv, nelt, nedge, coor, edge, conn, nvfix, numfix, itrace,
                    0, nqint, &fail);
  if (fail.code == NE_NOERROR)
    {
      if (pmesh[0] == 'N')
        {
          printf("\n The complete smoothed 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_smooth (d06cac).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

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

  return exit_status;
}
```

10.2 Program Data

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

10.3 Program Results

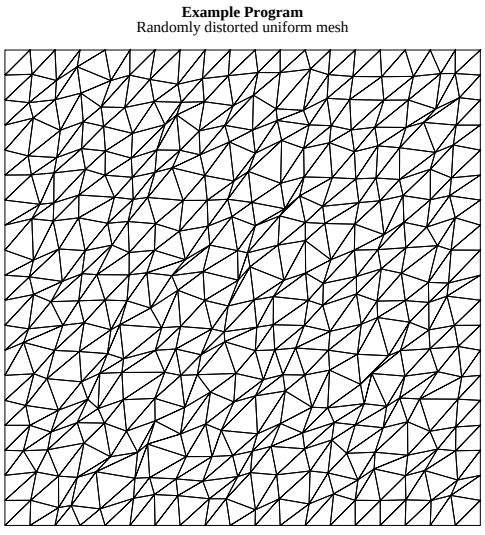
```
 nag_mesh2d_smooth (d06cac) Example Program Results

 The complete distorted mesh characteristics
 nv   =   400
 nelt =   722

 BEFORE SMOOTHING
 Minimum smoothness measure:       1.0144422
 Maximum smoothness measure:       6.2736750
 Distribution interval            Number of elements
      1.0144422 -       1.5403655       437
      1.5403655 -       2.0662888       191
      2.0662888 -       2.5922121        52
      2.5922121 -       3.1181354        17
      3.1181354 -       3.6440586        10
      3.6440586 -       4.1699819         6
      4.1699819 -       4.6959052         2
      4.6959052 -       5.2218285         4
      5.2218285 -       5.7477518         0
      5.7477518 -       6.2736750         2

 AFTER SMOOTHING
 Minimum smoothness measure:       1.3363949
 Maximum smoothness measure:       1.4744102
 Distribution interval            Number of elements
      1.3363949 -       1.3501964        15
      1.3501964 -       1.3639980        43
      1.3639980 -       1.3777995        69
      1.3777995 -       1.3916010       173
      1.3916010 -       1.4054025       232
      1.4054025 -       1.4192041       118
      1.4192041 -       1.4330056        52
      1.4330056 -       1.4468071         7
      1.4468071 -       1.4606086         8
      1.4606086 -       1.4744102         4

 The complete smoothed mesh characteristics
 nv   =   400
 nelt =   722
```



Distorted mesh smoothed and a uniform mesh recovered