

NAG Library Function Document

nag_mesh2d_join (d06dbc)

1 Purpose

nag_mesh2d_join (d06dbc) joins together (restitches) two adjacent, or overlapping, meshes.

2 Specification

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

void nag_mesh2d_join (double eps, Integer nv1, Integer nelt1, Integer nedge1,
  const double coor1[], const Integer edge1[], const Integer conn1[],
  const Integer reft1[], Integer nv2, Integer nelt2, Integer nedge2,
  const double coor2[], const Integer edge2[], const Integer conn2[],
  const Integer reft2[], Integer *nv3, Integer *nelt3, Integer *nedge3,
  double coor3[], Integer edge3[], Integer conn3[], Integer reft3[],
  Integer itrace, const char *outfile, NagError *fail)
```

3 Description

nag_mesh2d_join (d06dbc) joins together two adjacent, or overlapping, meshes. If the two meshes are adjacent then vertices belonging to the part of the boundary forming the common interface should coincide. If the two meshes overlap then vertices and triangles in the overlapping zone should coincide too.

This function is partly derived from material in the MODULEF package from INRIA (Institut National de Recherche en Informatique et Automatique).

4 References

None.

5 Arguments

- | | | |
|----|---|--------------|
| 1: | eps – double | <i>Input</i> |
| | <i>On entry:</i> the relative precision of the restitching of the two input meshes (see Section 9). | |
| | <i>Suggested value:</i> 0.001. | |
| | <i>Constraint:</i> eps > 0.0. | |
| 2: | nv1 – Integer | <i>Input</i> |
| | <i>On entry:</i> the total number of vertices in the first input mesh. | |
| | <i>Constraint:</i> nv1 ≥ 3. | |
| 3: | nelt1 – Integer | <i>Input</i> |
| | <i>On entry:</i> the number of triangular elements in the first input mesh. | |
| | <i>Constraint:</i> nelt1 ≤ 2 × nv1 – 1. | |

- 4: **nedge1** – Integer *Input*
On entry: the number of boundary edges in the first input mesh.
Constraint: **nedge1** ≥ 1 .
- 5: **coor1**[2 \times **nv1**] – const double *Input*
Note: the (i, j) th element of the matrix is stored in **coor1**[($j - 1$) \times 2 + $i - 1$].
On entry: **coor1**[($i - 1$) \times 2] contains the x coordinate of the i th vertex of the first input mesh, for $i = 1, 2, \dots, \mathbf{nv1}$; while **coor1**[($i - 1$) \times 2 + 1] contains the corresponding y coordinate.
- 6: **edge1**[3 \times **nedge1**] – const Integer *Input*
Note: the (i, j) th element of the matrix is stored in **edge1**[($j - 1$) \times 3 + $i - 1$].
On entry: the specification of the boundary edges of the first input mesh. **edge1**[($j - 1$) \times 3] and **edge1**[($j - 1$) \times 3 + 1] contain the vertex numbers of the two end points of the j th boundary edge. **edge1**[($j - 1$) \times 3 + 2] is a user-supplied tag for the j th boundary edge.
Constraint: $1 \leq \mathbf{edge1}[(j - 1) \times 3 + i - 1] \leq \mathbf{nv1}$ and **edge1**[($j - 1$) \times 3] \neq **edge1**[($j - 1$) \times 3 + 1], for $i = 1, 2$ and $j = 1, 2, \dots, \mathbf{nedge1}$.
- 7: **conn1**[3 \times **nelt1**] – const Integer *Input*
Note: the (i, j) th element of the matrix is stored in **conn1**[($j - 1$) \times 3 + $i - 1$].
On entry: the connectivity between triangles and vertices of the first input mesh. For each triangle j , **conn1**[($j - 1$) \times 3 + $i - 1$] gives the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \dots, \mathbf{nelt1}$.
Constraints:
 $1 \leq \mathbf{conn1}[(j - 1) \times 3 + i - 1] \leq \mathbf{nv1}$;
conn1[($j - 1$) \times 3] \neq **conn1**[($j - 1$) \times 3 + 1];
conn1[($j - 1$) \times 3] \neq **conn1**[($j - 1$) \times 3 + 2] and
conn1[($j - 1$) \times 3 + 1] \neq **conn1**[($j - 1$) \times 3 + 2], for $i = 1, 2, 3$ and $j = 1, 2, \dots, \mathbf{nelt1}$.
- 8: **reft1**[**nelt1**] – const Integer *Input*
On entry: **reft1**[$k - 1$] contains the user-supplied tag of the k th triangle from the first input mesh, for $k = 1, 2, \dots, \mathbf{nelt1}$.
- 9: **nv2** – Integer *Input*
On entry: the total number of vertices in the second input mesh.
Constraint: **nv2** ≥ 3 .
- 10: **nelt2** – Integer *Input*
On entry: the number of triangular elements in the second input mesh.
Constraint: **nelt2** $\leq 2 \times \mathbf{nv2} - 1$.
- 11: **nedge2** – Integer *Input*
On entry: the number of boundary edges in the second input mesh.
Constraint: **nedge2** ≥ 1 .
- 12: **coor2**[2 \times **nv2**] – const double *Input*
Note: the (i, j) th element of the matrix is stored in **coor2**[($j - 1$) \times 2 + $i - 1$].
On entry: **coor2**[($i - 1$) \times 2] contains the x coordinate of the i th vertex of the second input mesh, for $i = 1, 2, \dots, \mathbf{nv2}$; while **coor2**[($i - 1$) \times 2 + 1] contains the corresponding y coordinate.

- 13: **edge2**[3 × **nedge2**] – const Integer *Input*
- Note:** the (i, j) th element of the matrix is stored in **edge2**[($j - 1$) × 3 + $i - 1$].
- On entry:* the specification of the boundary edges of the second input mesh. **edge2**[($j - 1$) × 3] and **edge2**[($j - 1$) × 3 + 1] contain the vertex numbers of the two end points of the j th boundary edge. **edge2**[($j - 1$) × 3 + 2] is a user-supplied tag for the j th boundary edge.
- Constraint:* $1 \leq \mathbf{edge2}[(j - 1) \times 3 + i - 1] \leq \mathbf{nv2}$ and **edge2**[($j - 1$) × 3] ≠ **edge2**[($j - 1$) × 3 + 1], for $i = 1, 2$ and $j = 1, 2, \dots, \mathbf{nedge2}$.
- 14: **conn2**[3 × **nelt2**] – const Integer *Input*
- Note:** the (i, j) th element of the matrix is stored in **conn2**[($j - 1$) × 3 + $i - 1$].
- On entry:* the connectivity between triangles and vertices of the second input mesh. For each triangle j , **conn2**[($j - 1$) × 3 + $i - 1$] gives the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \dots, \mathbf{nelt2}$.
- Constraints:*
- $$1 \leq \mathbf{conn2}[(j - 1) \times 3 + i - 1] \leq \mathbf{nv2};$$
- $$\mathbf{conn2}[(j - 1) \times 3] \neq \mathbf{conn2}[(j - 1) \times 3 + 1];$$
- $$\mathbf{conn2}[(j - 1) \times 3] \neq \mathbf{conn2}[(j - 1) \times 3 + 2] \text{ and}$$
- $$\mathbf{conn2}[(j - 1) \times 3 + 1] \neq \mathbf{conn2}[(j - 1) \times 3 + 2], \text{ for } i = 1, 2, 3 \text{ and } j = 1, 2, \dots, \mathbf{nelt2}.$$
- 15: **reft2**[**nelt2**] – const Integer *Input*
- On entry:* **reft2**[$k - 1$] contains the user-supplied tag of the k th triangle from the second input mesh, for $k = 1, 2, \dots, \mathbf{nelt2}$.
- 16: **nv3** – Integer * *Output*
- On exit:* the total number of vertices in the resulting mesh.
- 17: **nelt3** – Integer * *Output*
- On exit:* the number of triangular elements in the resulting mesh.
- 18: **nedge3** – Integer * *Output*
- On exit:* the number of boundary edges in the resulting mesh.
- 19: **coor3**[dim] – double *Output*
- Note:** the dimension, dim , of the array **coor3** must be at least $2 \times (\mathbf{nv1} + \mathbf{nv2})$.
- The (i, j) th element of the matrix is stored in **coor3**[($j - 1$) × 2 + $i - 1$].
- On exit:* **coor3**[($i - 1$) × 2] will contain the x coordinate of the i th vertex of the resulting mesh, for $i = 1, 2, \dots, \mathbf{nv3}$; while **coor3**[($i - 1$) × 2 + 1] will contain the corresponding y coordinate.
- 20: **edge3**[dim] – Integer *Output*
- Note:** the dimension, dim , of the array **edge3** must be at least $3 \times (\mathbf{nedge1} + \mathbf{nedge2})$. This may be reduced to **nedge3** once that value is known.
- The (i, j) th element of the matrix is stored in **edge3**[($j - 1$) × 3 + $i - 1$].
- On exit:* the specification of the boundary edges of the resulting mesh. **edge3**[($j - 1$) × 3 + $i - 1$] will contain the vertex number of the i th end point ($i = 1, 2$) of the j th boundary or interface edge.
- If the two meshes overlap, **edge3**[($j - 1$) × 3 + 2] will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.
- If the two meshes are adjacent,

- (i) if the j th edge is part of the partition interface, then **edge3** $[(j-1) \times 3 + 2]$ will contain the value $1000 \times k_1 + k_2$ where k_1 and k_2 are the tags for the same edge of the first and the second mesh respectively;
- (ii) otherwise, **edge3** $[(j-1) \times 3 + 2]$ will contain the same tag as the corresponding edge belonging to the first and/or the second input mesh.

21: **conn3** $[dim]$ – Integer *Output*

Note: the dimension, dim , of the array **conn3** must be at least $3 \times (\mathbf{nelt1} + \mathbf{nelt2})$. This may be reduced to **nelt3** once that value is known.

The (i, j) th element of the matrix is stored in **conn3** $[(j-1) \times 3 + i - 1]$.

On exit: the connectivity between triangles and vertices of the resulting mesh. **conn3** $[(j-1) \times 3 + i - 1]$ will give the indices of its three vertices (in anticlockwise order), for $i = 1, 2, 3$ and $j = 1, 2, \dots, \mathbf{nelt3}$.

22: **ref3** $[dim]$ – Integer *Output*

Note: the dimension, dim , of the array **ref3** must be at least $\mathbf{nelt1} + \mathbf{nelt2}$. This may be reduced to **nelt3** once that value is known.

On exit: if the two meshes form a partition, **ref3** $[k-1]$ will contain the same tag as the corresponding triangle belonging to the first or the second input mesh, for $k = 1, 2, \dots, \mathbf{nelt3}$. If the two meshes overlap, then **ref3** $[k-1]$ will contain the value $1000 \times k_1 + k_2$ where k_1 and k_2 are the user-supplied tags for the same triangle of the first and the second mesh respectively, for $k = 1, 2, \dots, \mathbf{nelt3}$.

23: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_join (d06dbc).

itrace ≤ 0

No output is generated.

itrace ≥ 1

Details about the common vertices, edges and triangles to both meshes are printed.

24: **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.

25: **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, **nedge1** = $\langle value \rangle$.

Constraint: **nedge1** ≥ 1 .

On entry, **nedge2** = $\langle value \rangle$.

Constraint: **nedge2** ≥ 1 .

On entry, **nv1** = $\langle value \rangle$.

Constraint: **nv1** ≥ 3 .

On entry, **nv2** = $\langle value \rangle$.

Constraint: **nv2** ≥ 3 .

NE_INT_2

On entry, **nelt1** = $\langle value \rangle$ and **nv1** = $\langle value \rangle$.

Constraint: **nelt1** $\leq 2 \times \mathbf{nv1} - 1$.

On entry, **nelt2** = $\langle value \rangle$ and **nv2** = $\langle value \rangle$.

Constraint: **nelt2** $\leq 2 \times \mathbf{nv2} - 1$.

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

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

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

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

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

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

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

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

NE_INT_4

On entry, **CONN1**(I, J) = $\langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and **nv1** = $\langle value \rangle$.

Constraint: **CONN1**(I, J) ≥ 1 and **CONN1**(I, J) $\leq \mathbf{nv1}$, where **CONN1**(I, J) denotes **conn1**[($J - 1$) $\times 3 + I - 1$].

On entry, **CONN2**(I, J) = $\langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and **nv2** = $\langle value \rangle$.

Constraint: **CONN2**(I, J) ≥ 1 and **CONN2**(I, J) $\leq \mathbf{nv2}$, where **CONN2**(I, J) denotes **conn2**[($J - 1$) $\times 3 + I - 1$].

On entry, **EDGE1**(I, J) = $\langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and **nv1** = $\langle value \rangle$.

Constraint: **EDGE1**(I, J) ≥ 1 and **EDGE1**(I, J) $\leq \mathbf{nv1}$, where **EDGE1**(I, J) denotes **edge1**[($J - 1$) $\times 3 + I - 1$].

On entry, **EDGE2**(I, J) = $\langle value \rangle$, $I = \langle value \rangle$, $J = \langle value \rangle$ and **nv2** = $\langle value \rangle$.

Constraint: **EDGE2**(I, J) ≥ 1 and **EDGE2**(I, J) $\leq \mathbf{nv2}$, where **EDGE2**(I, J) denotes **edge2**[($J - 1$) $\times 3 + I - 1$].

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

A serious error has occurred in an internal call to the restitching routine. Check the input of the two meshes, especially the edges/vertices and/or the triangles/vertices connectivities. Seek expert help.

The function has detected a different number of coincident edges from the two meshes on the partition interface $\langle value \rangle$ $\langle value \rangle$. Check the input of the two meshes, especially the edges/vertices connectivity.

The function has detected a different number of coincident triangles from the two meshes in the overlapping zone $\langle value \rangle$ $\langle value \rangle$. Check the input of the two meshes, especially the triangles/vertices connectivity.

The function has detected only $\langle value \rangle$ coincident vertices with a precision $\mathbf{eps} = \langle value \rangle$. Either \mathbf{eps} should be changed or the two meshes are not restitchable.

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

On entry, $\mathbf{eps} = \langle value \rangle$.
Constraint: $\mathbf{eps} > 0.0$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

`nag_mesh2d_join` (d06dbc) finds all the common vertices between the two input meshes using the relative precision of the restitching argument \mathbf{eps} . You are advised to vary the value of \mathbf{eps} in the neighbourhood of 0.001 with $\mathbf{itrace} \geq 1$ to get the optimal value for the meshes under consideration.

10 Example

For this function two examples are presented. There is a single example program for `nag_mesh2d_join` (d06dbc), with a main program and the code to solve the two example problems given in Example 1 (ex1) and Example 2 (ex2).

Example 1 (ex1)

This example involves the unit square $[0,1]^2$ meshed uniformly, and then translated by a vector $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ (using `nag_mesh2d_trans` (d06dac)). This translated mesh is then restitched with the original mesh. Two cases are considered:

- (a) overlapping meshes ($u_1 = 15.0$, $u_2 = 17.0$),
- (b) partitioned meshes ($u_1 = 19.0$, $u_2 = 0.0$).

The mesh on the unit square has 400 vertices, 722 triangles and its boundary has 76 edges. In the partitioned case the resulting geometry is shown in Figure 1 in Section 10.3 while the restitched mesh is shown in Figure 2 in Section 10.3. In the overlapping case the geometry and mesh are shown in Figure 3 and Figure 4 in Section 10.3.

Example 2 (ex2)

This example restitches three geometries by calling the function `nag_mesh2d_join` (d06dbc) twice. The result is a mesh with three partitions. The first geometry is meshed by the Delaunay–Voronoi process (using `nag_mesh2d_delaunay` (d06abc)), the second one meshed by an Advancing Front algorithm (using `nag_mesh2d_front` (d06acc)), while the third one is the result of a rotation (by $-\pi/2$) of the second one (using `nag_mesh2d_trans` (d06dac)). The resulting geometry is shown in Figure 5 in Section 10.3 and restitched mesh in Figure 6 in Section 10.3.

10.1 Program Text

```

/* nag_mesh2d_join (d06dbc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 7, 2001.
 * Mark 7b revised, 2004.
 */

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

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

static int ex1(void);
static int ex2(void);

#define EDGE1(I, J)  edge1[3*((J) -1)+(I) -1]
#define EDGE2(I, J)  edge2[3*((J) -1)+(I) -1]
#define EDGE3(I, J)  edge3[3*((J) -1)+(I) -1]
#define EDGE4(I, J)  edge4[3*((J) -1)+(I) -1]
#define EDGE5(I, J)  edge5[3*((J) -1)+(I) -1]
#define CONN1(I, J)  conn1[3*((J) -1)+(I) -1]
#define CONN2(I, J)  conn2[3*((J) -1)+(I) -1]
#define CONN3(I, J)  conn3[3*((J) -1)+(I) -1]
#define CONN4(I, J)  conn4[3*((J) -1)+(I) -1]
#define CONN5(I, J)  conn5[3*((J) -1)+(I) -1]
#define COOR1(I, J)  coor1[2*((J) -1)+(I) -1]
#define COOR2(I, J)  coor2[2*((J) -1)+(I) -1]
#define COOR3(I, J)  coor3[2*((J) -1)+(I) -1]
#define COOR4(I, J)  coor4[2*((J) -1)+(I) -1]
#define COOR5(I, J)  coor5[2*((J) -1)+(I) -1]
#define TRANS(I, J)  trans[6*((J) -1)+(I) -1]
#define LINED(I, J)  lined[4*((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)
{
    Integer  exit_status_ex1 = 0;
    Integer  exit_status_ex2 = 0;

```

```

printf("nag_mesh2d_join (d06dbc) Example Program Results\n");
exit_status_ex1 = ex1();
exit_status_ex2 = ex2();

return (exit_status_ex1 == 0 && exit_status_ex2 == 0)? 0 : 1;
}

int ex1(void)
{
  const Integer nvmax = 900, nedmx = 200, neltmx = 2*nvmax+5, ntrans = 1,
              mode = 0;
  double      eps;
  Integer      exit_status, i, imax, itrace, itrans, jmax, jtrans, k, ktrans;
  Integer      nedgel, nedge2, nedge3, nelt1, nelt2, nelt3, nv1, nv2, nv3,
              reftk;
  Integer      imaxml, jmaxml, ind;
  char         pmesh[2];
  double       *coor1 = 0, *coor2 = 0, *coor3 = 0, *trans = 0;
  Integer      *conn1 = 0, *conn2 = 0, *conn3 = 0, *edge1 = 0, *edge2 = 0;
  Integer      *edge3 = 0, *itype = 0, *reft1 = 0, *reft2 = 0, *reft3 = 0;
  NagError     fail;

  INIT_FAIL(fail);
  exit_status = 0;

  printf("\nExample 1\n\n");

  /* Skip heading in data file */

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

  /* Read the mesh: coordinates and connectivity of the 1st domain */

#ifdef _WIN32
  scanf_s("%NAG_IFMT", &nv1);
#else
  scanf("%NAG_IFMT", &nv1);
#endif
#ifdef _WIN32
  scanf_s("%NAG_IFMT", &nelt1);
#else
  scanf("%NAG_IFMT", &nelt1);
#endif
#ifdef _WIN32
  scanf_s("%*[\n] ");
#else
  scanf("%*[\n] ");
#endif

  nv2 = nv1;
  nelt2 = nelt1;

  imax = 20;
  jmax = imax;
  imaxml = imax - 1;
  jmaxml = jmax - 1;
  nedgel = 2*(imaxml + jmaxml);
  nedge2 = nedgel;

  /* Allocate memory */

  if (!(coor1 = NAG_ALLOC(2*nv1, double)) ||

```



```

!(coor2 = NAG_ALLOC(2*nv2, double)) ||
!(coor3 = NAG_ALLOC(2*nvmax, double)) ||
!(trans = NAG_ALLOC(6*ntrans, double)) ||
!(conn1 = NAG_ALLOC(3*nelt1, Integer)) ||
!(conn2 = NAG_ALLOC(3*nelt2, Integer)) ||
!(conn3 = NAG_ALLOC(3*neltmx, Integer)) ||
!(edge1 = NAG_ALLOC(3*nedge1, Integer)) ||
!(edge2 = NAG_ALLOC(3*nedge2, Integer)) ||
!(edge3 = NAG_ALLOC(3*nedmx, Integer)) ||
!(itype = NAG_ALLOC(ntrans, Integer)) ||
!(reft1 = NAG_ALLOC(nelt1, Integer)) ||
!(reft2 = NAG_ALLOC(nelt2, Integer)) ||
!(reft3 = NAG_ALLOC(neltmx, Integer))
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}

for (i = 1; i <= nv1; ++i)
{
#ifdef _WIN32
scanf_s("%lf", &COOR1(1, i));
#else
scanf("%lf", &COOR1(1, i));
#endif
#ifdef _WIN32
scanf_s("%lf", &COOR1(2, i));
#else
scanf("%lf", &COOR1(2, i));
#endif
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif
}

for (k = 1; k <= nelt1; ++k)
{
#ifdef _WIN32
scanf_s("%"NAG_IFMT"", &CONN1(1, k));
#else
scanf("%"NAG_IFMT"", &CONN1(1, k));
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"", &CONN1(2, k));
#else
scanf("%"NAG_IFMT"", &CONN1(2, k));
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"", &CONN1(3, k));
#else
scanf("%"NAG_IFMT"", &CONN1(3, k));
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"", &reftk);
#else
scanf("%"NAG_IFMT"", &reftk);
#endif
#ifdef _WIN32
scanf_s("%*[\n] ");
#else
scanf("%*[\n] ");
#endif

reft1[k-1] = 1;
reft2[k-1] = 2;
}

#ifdef _WIN32

```

```

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

/* the edges of the boundary */

ind = 0;

for (i = 1; i <= imaxml; ++i)
{
++ind;
EDGE1(1, ind) = i;
EDGE1(2, ind) = i + 1;
EDGE1(3, ind) = 1;
}

for (i = 1; i <= jmaxml; ++i)
{
++ind;
EDGE1(1, ind) = i*imax;
EDGE1(2, ind) = (i+1)*imax;
EDGE1(3, ind) = 1;
}

for (i = 1; i <= imaxml; ++i)
{
++ind;
EDGE1(1, ind) = imax*jmax - i + 1;
EDGE1(2, ind) = imax*jmax - i;
EDGE1(3, ind) = 1;
}

for (i = 1; i <= jmaxml; ++i)
{
++ind;
EDGE1(1, ind) = (jmax - i)*imax + 1;
EDGE1(2, ind) = (jmax - i - 1)*imax + 1;
EDGE1(3, ind) = 1;
}

for (ktrans = 0; ktrans < 2; ++ktrans)
{
/* Translation of the 1st domain to obtain the 2nd domain */
/* KTRANS = 0 leading to a domain overlapping */
/* KTRANS = 1 leading to a domain partition */

if (ktrans == 0)
{
itrans = imax - 5;
jtrans = jmax - 3;
}
else
{
itrans = imax - 1;
jtrans = 0;
}

itype[0] = 1;
TRANS(1, 1) = (double) itrans/(imax - 1.0);
TRANS(2, 1) = (double) jtrans/(jmax - 1.0);
itrace = 0;

/* nag_mesh2d_trans (d06dac).
* Generates a mesh resulting from an affine transformation
* of a given mesh

```

```

*/
nag_mesh2d_trans(mode, nv2, nedge2, nelt2, ntrans, itype, trans, coor1,
                 edge1, conn1, coor2, edge2, conn2, itrace, 0,
                 &fail);
if (fail.code == NE_NOERROR)
{
  for (i = 1; i <= nedge2; ++i) EDGE2(3, i) = 2;

  /* Call to the restitching driver */

  itrace = 0;
  eps = 0.01;

  /* nag_mesh2d_join (d06dbc).
   * Joins together two given adjacent (possibly overlapping)
   * meshes
   */
  nag_mesh2d_join(eps, nv1, nelt1, nedge1, coor1, edge1, conn1, reft1,
                 nv2, nelt2, nedge2, coor2, edge2, conn2, reft2, &nv3,
                 &nelt3, &nedge3, coor3, edge3, conn3, reft3, itrace,
                 0, &fail);
  if (fail.code == NE_NOERROR)
  {
    if (pmesh[0] == 'N')
    {
      if (ktrans == 0)
      {
        printf("The restitched mesh characteristics\n");
        printf("in the overlapping case\n");
      }
      else
      {
        printf("in the partition case\n");
      }
      printf(" nv      =%6"NAG_IFMT"\n", nv3);
      printf(" nelt   =%6"NAG_IFMT"\n", nelt3);
      printf(" nedge  =%6"NAG_IFMT"\n", nedge3);
    }
    else if (pmesh[0] == 'Y')
    {
      /* Output the mesh to view it using the
       NAG Graphics Library */

      printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\n", nv3,
             nelt3, nedge3);

      for (i = 1; i <= nv3; ++i)
        printf(" %15.6e %15.6e\n",
              COOR3(1, i), COOR3(2, i));

      for (k = 1; k <= nelt3; ++k)
        printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"
              "%10"NAG_IFMT"\n", CONN3(1, k), CONN3(2, k),
              CONN3(3, k), reft3[k - 1]);

      for (k = 1; k <= nedge3; ++k)
        printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\n",
              EDGE3(1, k), EDGE3(2, k), EDGE3(3, k));
    }
    else
    {
      printf("Problem with the printing option Y or N\n");
      exit_status = -1;
      goto END;
    }
  }
  else
  {
    printf("Error from nag_mesh2d_join (d06dbc).\n%s\n",
          fail.message);
    exit_status = 1;
  }
}

```

```

        goto END;
    }
}
else
{
    printf("Error from nag_mesh2d_trans (d06dac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}
}

END:
NAG_FREE(coor1);
NAG_FREE(coor2);
NAG_FREE(coor3);
NAG_FREE(trans);
NAG_FREE(conn1);
NAG_FREE(conn2);
NAG_FREE(conn3);
NAG_FREE(edge1);
NAG_FREE(edge2);
NAG_FREE(edge3);
NAG_FREE(itype);
NAG_FREE(ref1);
NAG_FREE(ref2);
NAG_FREE(ref3);

return exit_status;
}

int ex2(void)
{
    const Integer nvmax = 900, nedmx = 200, neltnx = 2*nvmax+5,
                 ntrans = 1, nus = 0, nvint = 0, nvfix = 0;
    static double ruser[1] = {-1.0};
    double
    eps;
    Integer
    exit_status, i, itrace, j, k, l, ncomp, nedge1, nedge2, nedge3;
    Integer
    nedge4, nedge5, nelt1, nelt2, nelt3, nelt4, nelt5, nlines;
    Integer
    npropa, nqint, nv1, nv2, nv3, nv4, nv5, nvb1, nvb2, mode;
    char
    pmesh[2];
    double
    *coor1 = 0, *coor2 = 0, *coor3 = 0, *coor4 = 0, *coor5 = 0;
    double
    *coorch = 0, *coorus = 0, *rate = 0, *trans = 0, *weight = 0;
    Integer
    *conn1 = 0, *conn2 = 0, *conn3 = 0, *conn4 = 0, *conn5 = 0;
    Integer
    *edge1 = 0, *edge2 = 0, *edge3 = 0, *edge4 = 0, *edge5 = 0;
    Integer
    *itype = 0, *lcomp = 0, *lined = 0, *nlcomp = 0, *numfix = 0;
    Integer
    *ref1 = 0, *ref2 = 0, *ref3 = 0, *ref4 = 0, *ref5 = 0;
    NagError
    fail;
    Nag_Comm
    comm;

    INIT_FAIL(fail);

    /* For communication with user-supplied functions: */
    comm.user = ruser;

    exit_status = 0;

    printf("\nExample 2\n\n");

    /* Skip heading in data file */

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

    /* Build the mesh of the 1st domain */
    /* Initialise boundary mesh inputs: */
    /* the number of line and of the 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 (!(coor1 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor2 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor3 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor4 = NAG_ALLOC(2*nvmax, double)) ||
        !(coor5 = NAG_ALLOC(2*nvmax, double)) ||
        !(coorch = NAG_ALLOC(2*nlines, double)) ||
        !(coorus = NAG_ALLOC(1, double)) ||
        !(rate = NAG_ALLOC(nlines, double)) ||
        !(trans = NAG_ALLOC(6*ntrans, double)) ||
        !(weight = NAG_ALLOC(1, double)) ||
        !(conn1 = NAG_ALLOC(3*neltmx, Integer)) ||
        !(conn2 = NAG_ALLOC(3*neltmx, Integer)) ||
        !(conn3 = NAG_ALLOC(3*neltmx, Integer)) ||
        !(conn4 = NAG_ALLOC(3*neltmx, Integer)) ||
        !(conn5 = NAG_ALLOC(3*neltmx, Integer)) ||
        !(edge1 = NAG_ALLOC(3*nedmx, Integer)) ||
        !(edge2 = NAG_ALLOC(3*nedmx, Integer)) ||
        !(edge3 = NAG_ALLOC(3*nedmx, Integer)) ||
        !(edge4 = NAG_ALLOC(3*nedmx, Integer)) ||
        !(edge5 = NAG_ALLOC(3*nedmx, Integer)) ||
        !(itype = NAG_ALLOC(ntrans, Integer)) ||
        !(lcomp = NAG_ALLOC(nlines, Integer)) ||
        !(lined = NAG_ALLOC(4*nlines, Integer)) ||
        !(numfix = NAG_ALLOC(1, Integer)) ||
        !(reft1 = NAG_ALLOC(2*nvmax+5, Integer)) ||
        !(reft2 = NAG_ALLOC(2*nvmax+5, Integer)) ||
        !(reft3 = NAG_ALLOC(2*nvmax+5, Integer)) ||
        !(reft4 = NAG_ALLOC(2*nvmax+5, Integer)) ||
        !(reft5 = NAG_ALLOC(2*nvmax+5, Integer)))
    {
        printf("Allocation failure\\n");
        exit_status = -1;
        goto END;
    }

    /* Characteristic points of the boundary geometry */

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

    for (j = 1; j <= nlines; ++j)
    {
#ifdef _WIN32
        scanf_s("%lf", &COORCH(2, j));
#else
        scanf("%lf", &COORCH(2, j));
#endif
    }

```

```

#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 */
    /* on the boundary and their data */

#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 = 1;

    for (i = 1; i <= ncomp; ++i)
    {
#ifdef _WIN32
        scanf_s("%NAG_IFMT", &nlcomp[i - 1]);
#else
        scanf("%NAG_IFMT", &nlcomp[i - 1]);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif
        l = j + abs(nlcomp[i - 1]) - 1;
#ifdef _WIN32
        for (k = j; k <= l; ++k) scanf_s("%NAG_IFMT", &lcomp[k - 1]);
#else
        for (k = j; k <= l; ++k) scanf("%NAG_IFMT", &lcomp[k - 1]);

```

```

#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
    j += abs(nlcomp[i - 1]);
}

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, &nvbl, coor1, &nedge1,
                edge1, itrace, 0, &comm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_bound (d06bac).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Generate mesh using Delaunay-Voronoi method */

/* Initialise mesh control parameters */

itrace = 0;
npropa = 1;

/* nag_mesh2d_delaunay (d06abc).
 * Generates a two-dimensional mesh using a Delaunay-Voronoi
 * process
 */
nag_mesh2d_delaunay(nvbl, nvint, nvmax, nedge1, edge1, &nv1, &nelt1, coor1,
                   conn1, weight, npropa, itrace, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_delaunay (d06abc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

for (k = 1; k <= nelt1; ++k) reft1[k - 1] = 1;

/* Call the smoothing routine */

nqint = 10;
/* nag_mesh2d_smooth (d06cac).
 * Uses a barycentering technique to smooth a given mesh
 */
nag_mesh2d_smooth(nv1, nelt1, nedge1, coor1, edge1, conn1, nvfix, numfix,
                 itrace, 0, nqint, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_smooth (d06cac).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Build the mesh of the 2nd domain */

#ifdef _WIN32
    scanf_s("%"NAG_IFMT"", &nlines);
#else

```

```

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

    /* Characteristic points of the boundary geometry */

#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 data */

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

    j = 1;
    for (i = 1; i <= ncomp; ++i)
    {
#ifdef _WIN32

```



```

        scanf_s("%NAG_IFMT", &nlcomp[i - 1]);
#else
        scanf("%NAG_IFMT", &nlcomp[i - 1]);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif

        for (k = j; k <= j+abs(nlcomp[i-1])-1; ++k)
#ifdef _WIN32
            scanf_s("%NAG_IFMT", &lcomp[k - 1]);
#else
            scanf("%NAG_IFMT", &lcomp[k - 1]);
#endif
#ifdef _WIN32
        scanf_s("%*[\n] ");
#else
        scanf("%*[\n] ");
#endif
        j += abs(nlcomp[i-1]);
    }

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

    itrace = 0;

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

    /* nag_mesh2d_bound (d06bac), see above. */
    nag_mesh2d_bound(nlines, coorch, lined, fbnd, coorus, nus, rate, ncomp,
                    nlcomp, lcomp, nvmax, nedmx, &nvb2, coor2, &nedge2, edge2,
                    itrace, 0, &comm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_mesh2d_bound (d06bac).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }

    /* Generate mesh using the advancing front method */

    itrace = 0;

    /* nag_mesh2d_front (d06acc).
     * Generates a two-dimensional mesh using an Advancing-front
     * method
     */
    nag_mesh2d_front(nvb2, nvint, nvmax, nedge2, edge2, &nv2, &nelt2, coor2,
                    conn2, weight, itrace, 0, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_mesh2d_front (d06acc).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }

    for (k = 1; k <= nelt2; ++k) reft2[k - 1] = 2;

```

```

/* Rotation of the 2nd domain mesh */
/* to produce the 3rd mesh domain */

itype[0] = 3;
TRANS(1, 1) = 6.0;
TRANS(2, 1) = -1.0;
TRANS(3, 1) = -90.0;
itrace = 0;
mode = 0;

/* nag_mesh2d_trans (d06dac), see above. */
nag_mesh2d_trans(mode, nv2, nedge2, nelt2, ntrans, itype, trans, coor2,
                edge2, conn2, coor3, edge3, conn3, itrace, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_trans (d06dac).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

nv3 = nv2;
nelt3 = nelt2;
nedge3 = nedge2;

for (k = 1; k <= nelt3; ++k) reft3[k - 1] = 3;

/* Restitching meshes 1 and 2 to form mesh 4 */

eps = 0.001;
itrace = 0;

/* nag_mesh2d_join (d06dbc), see above. */
nag_mesh2d_join(eps, nv1, nelt1, nedge1, coor1, edge1, conn1, reft1, nv2,
                nelt2, nedge2, coor2, edge2, conn2, reft2, &nv4, &nelt4,
                &nedge4, coor4, edge4, conn4, reft4, itrace, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_join (d06dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

/* Restitching meshes 3 and 4 to form mesh 5 */

itrace = 0;

/* nag_mesh2d_join (d06dbc), see above. */
nag_mesh2d_join(eps, nv4, nelt4, nedge4, coor4, edge4, conn4, reft4, nv3,
                nelt3, nedge3, coor3, edge3, conn3, reft3, &nv5, &nelt5,
                &nedge5, coor5, edge5, conn5, reft5, itrace, 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_mesh2d_join (d06dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

if (pmesh[0] == 'N')
{
    printf("The restitched mesh characteristics\n");
    printf(" nv      =%6"NAG_IFMT"\n", nv5);
    printf(" nelt   =%6"NAG_IFMT"\n", nelt5);
    printf(" nedge  =%6"NAG_IFMT"\n", nedge5);
}
else if (pmesh[0] == 'Y')
{
    /* Output the mesh to view it using the NAG Graphics Library */

```

```

    printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\n", nv5, nelt5, nedge5);

    for (i = 1; i <= nv5; ++i)
        printf(" %15.6e %15.6e\n",
            COOR5(1, i), COOR5(2, i));

    for (k = 1; k <= nelt5; ++k)
        printf("%10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\n",
            CONN5(1, k), CONN5(2, k), CONN5(3, k), reft5[k - 1]);

    for (k = 1; k <= nedge5; ++k)
        printf(" %10"NAG_IFMT"%10"NAG_IFMT"%10"NAG_IFMT"\n",
            EDGE5(1, k), EDGE5(2, k), EDGE5(3, k));
}
else
{
    printf("Problem with the printing option Y or N\n");
}
}

END:
NAG_FREE(coor1);
NAG_FREE(coor2);
NAG_FREE(coor3);
NAG_FREE(coor4);
NAG_FREE(coor5);
NAG_FREE(coorch);
NAG_FREE(coorus);
NAG_FREE(rate);
NAG_FREE(trans);
NAG_FREE(weight);
NAG_FREE(conn1);
NAG_FREE(conn2);
NAG_FREE(conn3);
NAG_FREE(conn4);
NAG_FREE(conn5);
NAG_FREE(edge1);
NAG_FREE(edge2);
NAG_FREE(edge3);
NAG_FREE(edge4);
NAG_FREE(edge5);
NAG_FREE(itype);
NAG_FREE(lcomp);
NAG_FREE(lined);
NAG_FREE(nlcomp);
NAG_FREE(numfix);
NAG_FREE(reft1);
NAG_FREE(reft2);
NAG_FREE(reft3);
NAG_FREE(reft4);
NAG_FREE(reft5);

return exit_status;
}

double NAG_CALL fbnd(Integer i, double x, double y, Nag_Comm *pcomm)
{
    double radius2, x0, y0, ret_val;

    if (pcomm->user[0] == -1.0)
    {
        printf("(User-supplied callback fbnd, first invocation.)\n");
        pcomm->user[0] = 0.0;
    }

    ret_val = 0.0;
    switch (i)
    {
        case 1:

            /* inner circle */

```

```

    x0 = 0.0;
    y0 = 0.0;
    radius2 = 1.0;
    ret_val = (x-x0)*(x-x0) + (y-y0)*(y-y0) - radius2;
    break;

case 2:

    /* outer circle */

    x0 = 0.0;
    y0 = 0.0;
    radius2 = 5.0;
    ret_val = (x-x0)*(x-x0) + (y-y0)*(y-y0) - radius2;
    break;
}

return ret_val;
}

```

10.2 Program Data

Note 1: since the data file for this example is quite large only a section of it is reproduced in this document. The full data file is distributed with your implementation.

nag_mesh2d_join (d06dbc) Example Program Data
Example 1

```

    400      722      :NV1 NELT1
0.000000E+00 0.000000E+00
0.526316E-01 0.000000E+00
0.105263E+00 0.000000E+00
0.157895E+00 0.000000E+00
0.210526E+00 0.000000E+00
0.263158E+00 0.000000E+00
0.315789E+00 0.000000E+00
0.368421E+00 0.000000E+00
0.421053E+00 0.000000E+00
0.473684E+00 0.000000E+00
0.526316E+00 0.000000E+00
0.578947E+00 0.000000E+00
0.631579E+00 0.000000E+00
0.684211E+00 0.000000E+00
0.736842E+00 0.000000E+00
0.789474E+00 0.000000E+00
0.842105E+00 0.000000E+00
0.894737E+00 0.000000E+00
0.947368E+00 0.000000E+00
0.100000E+01 0.000000E+00
0.000000E+00 0.526316E-01
0.530040E-01 0.529717E-01
0.106007E+00 0.533597E-01
0.159150E+00 0.534245E-01
0.209592E+00 0.531136E-01
0.260599E+00 0.529632E-01
0.313038E+00 0.533547E-01
0.366919E+00 0.539130E-01
0.420609E+00 0.534341E-01
0.473520E+00 0.531335E-01
0.526337E+00 0.532672E-01
0.579088E+00 0.529351E-01
0.631888E+00 0.527519E-01
0.684651E+00 0.525232E-01
0.737256E+00 0.521887E-01
0.789430E+00 0.514365E-01
0.841553E+00 0.511852E-01
0.894133E+00 0.513826E-01
0.947067E+00 0.520693E-01
0.100000E+01 0.526316E-01
0.000000E+00 0.105263E+00

```

0.534333E-01	0.105629E+00
0.106235E+00	0.106414E+00
0.159956E+00	0.107543E+00
0.209996E+00	0.107282E+00
0.259977E+00	0.106009E+00
0.311401E+00	0.105685E+00
0.365802E+00	0.106250E+00
0.420642E+00	0.106115E+00
0.473410E+00	0.105494E+00
0.526068E+00	0.105215E+00
0.578675E+00	0.105602E+00
0.631648E+00	0.105363E+00
0.684596E+00	0.104955E+00
0.737448E+00	0.104762E+00
0.789820E+00	0.104321E+00
0.841780E+00	0.103634E+00
0.893995E+00	0.103504E+00
0.946852E+00	0.104258E+00
0.100000E+01	0.105263E+00
0.000000E+00	0.157895E+00
0.538039E-01	0.157947E+00
0.106545E+00	0.158585E+00
0.158858E+00	0.160703E+00
0.210128E+00	0.161508E+00
0.260398E+00	0.160854E+00
0.310943E+00	0.159774E+00
0.363600E+00	0.159143E+00
0.419451E+00	0.159396E+00
0.472928E+00	0.158225E+00
0.525577E+00	0.157661E+00
0.578223E+00	0.157998E+00
0.631075E+00	0.158158E+00
0.683956E+00	0.157847E+00
0.736875E+00	0.157691E+00
0.789720E+00	0.157682E+00
0.842080E+00	0.157078E+00
0.894374E+00	0.156562E+00
0.946955E+00	0.156934E+00
0.100000E+01	0.157895E+00
0.000000E+00	0.210526E+00
0.537515E-01	0.209851E+00
0.106776E+00	0.209921E+00
0.158931E+00	0.211672E+00
0.210208E+00	0.214046E+00
0.260742E+00	0.214868E+00
0.311521E+00	0.214591E+00
0.363876E+00	0.214038E+00
0.418145E+00	0.212959E+00
0.472230E+00	0.211629E+00
0.525111E+00	0.210427E+00
0.577985E+00	0.210469E+00
0.630736E+00	0.210820E+00
0.683370E+00	0.210792E+00
0.735968E+00	0.210731E+00
0.788992E+00	0.210732E+00
0.841819E+00	0.210398E+00
0.894612E+00	0.209890E+00
0.947184E+00	0.209907E+00
0.100000E+01	0.210526E+00
0.000000E+00	0.263158E+00
0.536725E-01	0.262201E+00
0.106635E+00	0.261527E+00
0.158876E+00	0.262359E+00
0.210666E+00	0.264518E+00
0.261283E+00	0.266476E+00
0.312324E+00	0.267273E+00
0.364660E+00	0.267047E+00
0.418511E+00	0.265891E+00
0.471999E+00	0.264455E+00
0.524845E+00	0.263507E+00
0.577755E+00	0.263408E+00

0.630640E+00	0.263527E+00
0.683254E+00	0.263258E+00
0.735891E+00	0.262664E+00
0.788569E+00	0.262596E+00
0.841356E+00	0.262854E+00
0.894369E+00	0.262874E+00
0.947225E+00	0.262874E+00
0.100000E+01	0.263158E+00
0.000000E+00	0.315789E+00
0.545160E-01	0.315767E+00
0.107159E+00	0.314677E+00
0.158994E+00	0.314120E+00
0.210434E+00	0.314624E+00
0.261605E+00	0.316212E+00
0.313007E+00	0.317562E+00
0.365417E+00	0.317948E+00
0.419192E+00	0.317475E+00
0.473235E+00	0.316759E+00
0.525468E+00	0.316165E+00
0.577946E+00	0.316445E+00
0.630796E+00	0.316505E+00
0.683633E+00	0.316034E+00
0.736328E+00	0.315338E+00
0.788940E+00	0.315112E+00
0.841110E+00	0.315404E+00
0.894159E+00	0.315900E+00
0.947129E+00	0.315842E+00
0.100000E+01	0.315789E+00
0.000000E+00	0.368421E+00
0.562040E-01	0.369910E+00
0.108298E+00	0.368859E+00
0.159916E+00	0.367246E+00
0.210824E+00	0.365954E+00
0.261646E+00	0.365335E+00
0.313118E+00	0.366449E+00
0.365457E+00	0.367228E+00
0.418953E+00	0.367797E+00
0.473433E+00	0.368303E+00
0.526137E+00	0.368453E+00
0.578680E+00	0.369026E+00
0.631340E+00	0.369213E+00
0.684293E+00	0.368774E+00
0.737208E+00	0.368247E+00
0.789620E+00	0.368658E+00
0.842011E+00	0.369427E+00
0.894600E+00	0.369468E+00
0.947268E+00	0.368954E+00
0.100000E+01	0.368421E+00
0.000000E+00	0.421053E+00
0.548594E-01	0.423169E+00
0.108224E+00	0.423363E+00
0.160449E+00	0.421555E+00
0.211961E+00	0.419389E+00
0.262781E+00	0.417386E+00
0.313498E+00	0.415919E+00
0.365485E+00	0.416287E+00
0.418574E+00	0.417343E+00
0.472328E+00	0.418262E+00
0.525775E+00	0.420205E+00
0.579114E+00	0.421262E+00
0.631883E+00	0.421460E+00
0.684610E+00	0.421103E+00
0.737445E+00	0.420855E+00
0.790017E+00	0.421458E+00
0.842418E+00	0.422405E+00
0.895069E+00	0.422629E+00
0.947591E+00	0.421965E+00
0.100000E+01	0.421053E+00
0.000000E+00	0.473684E+00
0.532319E-01	0.475673E+00
0.106492E+00	0.477027E+00

0.159501E+00	0.476105E+00
0.211988E+00	0.474016E+00
0.263778E+00	0.471957E+00
0.315022E+00	0.470082E+00
0.366430E+00	0.468910E+00
0.418881E+00	0.469108E+00
0.471646E+00	0.469380E+00
0.524570E+00	0.471703E+00
0.578628E+00	0.473569E+00
0.632170E+00	0.473634E+00
0.684783E+00	0.473058E+00
0.736978E+00	0.472478E+00
0.789629E+00	0.472845E+00
0.842153E+00	0.473917E+00
0.894842E+00	0.474621E+00
0.947590E+00	0.474445E+00
0.100000E+01	0.473684E+00
0.000000E+00	0.526316E+00
0.514925E-01	0.527735E+00
0.104211E+00	0.529574E+00
0.157661E+00	0.529770E+00
0.211100E+00	0.528675E+00
0.263720E+00	0.527155E+00
0.316379E+00	0.526398E+00
0.368652E+00	0.525467E+00
0.420519E+00	0.525018E+00
0.472412E+00	0.525035E+00
0.524455E+00	0.525557E+00
0.577960E+00	0.526664E+00
0.631985E+00	0.526440E+00
0.684635E+00	0.525219E+00
0.736406E+00	0.524028E+00
0.788965E+00	0.523759E+00
0.841752E+00	0.524678E+00
0.894427E+00	0.525944E+00
0.947283E+00	0.526498E+00
0.100000E+01	0.526316E+00
0.000000E+00	0.578947E+00
0.508223E-01	0.580673E+00
0.102950E+00	0.582013E+00
0.156261E+00	0.582639E+00
0.209694E+00	0.582137E+00
0.262775E+00	0.581621E+00
0.316044E+00	0.582413E+00
0.370283E+00	0.581142E+00
0.423492E+00	0.582131E+00
0.474535E+00	0.581193E+00
0.526077E+00	0.580824E+00
0.578169E+00	0.580756E+00
0.631393E+00	0.580005E+00
0.684214E+00	0.578309E+00
0.736055E+00	0.576460E+00
0.788503E+00	0.575477E+00
0.841652E+00	0.576006E+00
0.894472E+00	0.577533E+00
0.947193E+00	0.578628E+00
0.100000E+01	0.578947E+00
0.000000E+00	0.631579E+00
0.515621E-01	0.633678E+00
0.103835E+00	0.634871E+00
0.156589E+00	0.635456E+00
0.209397E+00	0.635196E+00
0.262134E+00	0.634720E+00
0.315575E+00	0.634618E+00
0.370645E+00	0.634457E+00
0.423719E+00	0.633784E+00
0.476017E+00	0.633788E+00
0.528540E+00	0.634864E+00
0.579371E+00	0.634164E+00
0.631164E+00	0.633554E+00
0.683326E+00	0.631992E+00

0.735292E+00	0.629577E+00
0.787892E+00	0.628376E+00
0.841453E+00	0.628291E+00
0.894691E+00	0.629566E+00
0.947366E+00	0.630913E+00
0.100000E+01	0.631579E+00
0.000000E+00	0.684211E+00
0.526674E-01	0.686277E+00
0.105418E+00	0.687895E+00
0.158169E+00	0.688395E+00
0.210509E+00	0.688004E+00
0.262606E+00	0.687309E+00
0.315494E+00	0.686923E+00
0.369671E+00	0.686428E+00
0.423157E+00	0.685926E+00
0.475555E+00	0.685169E+00
0.528315E+00	0.685377E+00
0.580571E+00	0.685916E+00
0.631576E+00	0.686228E+00
0.682963E+00	0.685657E+00
0.734669E+00	0.684106E+00
0.786560E+00	0.682127E+00
0.840591E+00	0.681347E+00
0.894361E+00	0.681853E+00
0.947369E+00	0.683167E+00
0.100000E+01	0.684211E+00
0.000000E+00	0.736842E+00
0.525525E-01	0.738402E+00
0.105929E+00	0.740228E+00
0.159138E+00	0.740783E+00
0.211875E+00	0.740282E+00
0.264131E+00	0.739646E+00
0.315782E+00	0.739109E+00
0.368611E+00	0.739363E+00
0.421782E+00	0.738853E+00
0.474160E+00	0.737399E+00
0.527515E+00	0.736712E+00
0.580596E+00	0.737286E+00
0.632354E+00	0.738184E+00
0.683394E+00	0.738540E+00
0.734750E+00	0.738167E+00
0.786534E+00	0.736760E+00
0.839112E+00	0.734838E+00
0.893426E+00	0.734706E+00
0.946996E+00	0.735569E+00
0.100000E+01	0.736842E+00
0.000000E+00	0.789474E+00
0.522839E-01	0.790785E+00
0.105635E+00	0.791683E+00
0.159138E+00	0.791887E+00
0.212240E+00	0.791595E+00
0.264920E+00	0.790667E+00
0.316755E+00	0.790307E+00
0.368556E+00	0.791156E+00
0.421044E+00	0.791569E+00
0.473635E+00	0.790315E+00
0.526484E+00	0.788660E+00
0.580123E+00	0.788450E+00
0.633694E+00	0.789734E+00
0.684453E+00	0.790438E+00
0.735240E+00	0.791083E+00
0.786532E+00	0.790796E+00
0.838996E+00	0.789559E+00
0.892616E+00	0.788465E+00
0.946507E+00	0.788501E+00
0.100000E+01	0.789474E+00
0.000000E+00	0.842105E+00
0.523473E-01	0.843080E+00
0.105941E+00	0.842714E+00
0.159189E+00	0.842502E+00
0.212004E+00	0.842294E+00

0.264979E+00	0.841753E+00
0.317598E+00	0.841354E+00
0.369415E+00	0.842226E+00
0.421163E+00	0.843557E+00
0.473189E+00	0.843214E+00
0.525571E+00	0.841287E+00
0.579046E+00	0.840113E+00
0.632734E+00	0.839827E+00
0.685273E+00	0.841863E+00
0.736184E+00	0.843417E+00
0.786892E+00	0.843934E+00
0.838821E+00	0.843293E+00
0.892179E+00	0.842328E+00
0.946154E+00	0.841808E+00
0.100000E+01	0.842105E+00
0.000000E+00	0.894737E+00
0.519324E-01	0.895315E+00
0.105695E+00	0.894921E+00
0.159166E+00	0.894230E+00
0.211854E+00	0.893718E+00
0.264323E+00	0.892842E+00
0.317522E+00	0.893157E+00
0.369989E+00	0.893822E+00
0.421780E+00	0.895319E+00
0.473248E+00	0.895979E+00
0.525260E+00	0.894688E+00
0.578264E+00	0.893115E+00
0.631788E+00	0.892853E+00
0.684946E+00	0.894285E+00
0.736760E+00	0.895964E+00
0.787742E+00	0.896536E+00
0.839280E+00	0.896130E+00
0.892136E+00	0.895464E+00
0.945949E+00	0.894935E+00
0.100000E+01	0.894737E+00
0.000000E+00	0.947368E+00
0.520265E-01	0.947550E+00
0.105228E+00	0.947497E+00
0.158898E+00	0.947109E+00
0.211410E+00	0.946602E+00
0.264014E+00	0.945894E+00
0.316723E+00	0.945999E+00
0.369490E+00	0.946458E+00
0.421818E+00	0.947325E+00
0.473756E+00	0.948140E+00
0.525704E+00	0.947794E+00
0.578290E+00	0.946822E+00
0.631358E+00	0.946342E+00
0.684441E+00	0.947048E+00
0.736995E+00	0.948310E+00
0.788488E+00	0.949111E+00
0.840525E+00	0.948452E+00
0.893076E+00	0.947997E+00
0.946255E+00	0.947654E+00
0.100000E+01	0.947368E+00
0.000000E+00	0.100000E+01
0.526316E-01	0.100000E+01
0.105263E+00	0.100000E+01
0.157895E+00	0.100000E+01
0.210526E+00	0.100000E+01
0.263158E+00	0.100000E+01
0.315789E+00	0.100000E+01
0.368421E+00	0.100000E+01
0.421053E+00	0.100000E+01
0.473684E+00	0.100000E+01
0.526316E+00	0.100000E+01
0.578947E+00	0.100000E+01
0.631579E+00	0.100000E+01
0.684211E+00	0.100000E+01
0.736842E+00	0.100000E+01
0.789474E+00	0.100000E+01

```

0.842105E+00  0.100000E+01
0.894737E+00  0.100000E+01
0.947368E+00  0.100000E+01
0.100000E+01  0.100000E+01      :COOR1(1:2,1:NV1)
   1           2           22           0
   1           22          21           0
   2           3           23           0
   2           23          22           0
   3           4           24           0
   3           24          23           0
   4           5           25           0
   4           25          24           0
   5           6           26           0
   5           26          25           0
   6           7           27           0
   6           27          26           0
   7           8           28           0
   7           28          27           0
   8           9           29           0
   8           29          28           0
   9           10          30           0
   9           30          29           0
  10           11          31           0
  10           31          30           0
  11           12          32           0
  11           32          31           0
  12           13          33           0
  12           33          32           0
  13           14          34           0
  13           34          33           0
  14           15          35           0
  14           35          34           0
  15           16          36           0
  15           36          35           0
  16           17          37           0
  16           37          36           0
  17           18          38           0
  17           38          37           0
  18           19          39           0
  18           39          38           0
  19           20          40           0
  19           40          39           0
  21           22          42           0
  21           42          41           0
  22           23          43           0
  22           43          42           0
  23           24          44           0
  23           44          43           0
  24           25          45           0
  24           45          44           0
  25           26          46           0
  25           46          45           0
  26           27          47           0
  26           47          46           0
  27           28          48           0
  27           48          47           0
  28           29          49           0
  28           49          48           0
  29           30          50           0
  29           50          49           0
  30           31          51           0
  30           51          50           0
  31           32          52           0
  31           52          51           0
  32           33          53           0
  32           53          52           0
  33           34          54           0
  33           54          53           0
  34           35          55           0
  34           55          54           0
  35           36          56           0

```

35	56	55	0
36	37	57	0
36	57	56	0
37	38	58	0
37	58	57	0
38	39	59	0
38	59	58	0
39	40	60	0
39	60	59	0
41	42	62	0
41	62	61	0
42	43	63	0
42	63	62	0
43	44	64	0
43	64	63	0
44	45	65	0
44	65	64	0
45	46	66	0
45	66	65	0
46	47	67	0
46	67	66	0
47	48	68	0
47	68	67	0
48	49	69	0
48	69	68	0
49	50	70	0
49	70	69	0
50	51	71	0
50	71	70	0
51	52	72	0
51	72	71	0
52	53	73	0
52	73	72	0
53	54	74	0
53	74	73	0
54	55	75	0
54	75	74	0
55	56	76	0
55	76	75	0
56	57	77	0
56	77	76	0
57	58	78	0
57	78	77	0
58	59	79	0
58	79	78	0
59	60	80	0
59	80	79	0
61	62	82	0
61	82	81	0
62	63	83	0
62	83	82	0
63	64	84	0
63	84	83	0
64	65	85	0
64	85	84	0
65	66	86	0
65	86	85	0
66	67	87	0
66	87	86	0
67	68	88	0
67	88	87	0
68	69	89	0
68	89	88	0
69	70	90	0
69	90	89	0
70	71	91	0
70	91	90	0
71	72	92	0
71	92	91	0
72	73	93	0
72	93	92	0

73	74	94	0
73	94	93	0
74	75	95	0
74	95	94	0
75	76	96	0
75	96	95	0
76	77	97	0
76	97	96	0
77	78	98	0
77	98	97	0
78	79	99	0
78	99	98	0
79	80	100	0
79	100	99	0
81	82	102	0
81	102	101	0
82	83	103	0
82	103	102	0
83	84	104	0
83	104	103	0
84	85	105	0
84	105	104	0
85	86	106	0
85	106	105	0
86	87	107	0
86	107	106	0
87	88	108	0
87	108	107	0
88	89	109	0
88	109	108	0
89	90	110	0
89	110	109	0
90	91	111	0
90	111	110	0
91	92	112	0
91	112	111	0
92	93	113	0
92	113	112	0
93	94	114	0
93	114	113	0
94	95	115	0
94	115	114	0
95	96	116	0
95	116	115	0
96	97	117	0
96	117	116	0
97	98	118	0
97	118	117	0
98	99	119	0
98	119	118	0
99	100	120	0
99	120	119	0
101	102	122	0
101	122	121	0
102	103	123	0
102	123	122	0
103	104	124	0
103	124	123	0
104	105	125	0
104	125	124	0
105	106	126	0
105	126	125	0
106	107	127	0
106	127	126	0
107	108	128	0
107	128	127	0
108	109	129	0
108	129	128	0
109	110	130	0
109	130	129	0
110	111	131	0

110	131	130	0
111	112	132	0
111	132	131	0
112	113	133	0
112	133	132	0
113	114	134	0
113	134	133	0
114	115	135	0
114	135	134	0
115	116	136	0
115	136	135	0
116	117	137	0
116	137	136	0
117	118	138	0
117	138	137	0
118	119	139	0
118	139	138	0
119	120	140	0
119	140	139	0
121	122	142	0
121	142	141	0
122	123	143	0
122	143	142	0
123	124	144	0
123	144	143	0
124	125	145	0
124	145	144	0
125	126	146	0
125	146	145	0
126	127	147	0
126	147	146	0
127	128	148	0
127	148	147	0
128	129	149	0
128	149	148	0
129	130	150	0
129	150	149	0
130	131	151	0
130	151	150	0
131	132	152	0
131	152	151	0
132	133	153	0
132	153	152	0
133	134	154	0
133	154	153	0
134	135	155	0
134	155	154	0
135	136	156	0
135	156	155	0
136	137	157	0
136	157	156	0
137	138	158	0
137	158	157	0
138	139	159	0
138	159	158	0
139	140	160	0
139	160	159	0
141	142	162	0
141	162	161	0
142	143	163	0
142	163	162	0
143	144	164	0
143	164	163	0
144	145	165	0
144	165	164	0
145	146	166	0
145	166	165	0
146	147	167	0
146	167	166	0
147	148	168	0
147	168	167	0

148	149	169	0
148	169	168	0
149	150	170	0
149	170	169	0
150	151	171	0
150	171	170	0
151	152	172	0
151	172	171	0
152	153	173	0
152	173	172	0
153	154	174	0
153	174	173	0
154	155	175	0
154	175	174	0
155	156	176	0
155	176	175	0
156	157	177	0
156	177	176	0
157	158	178	0
157	178	177	0
158	159	179	0
158	179	178	0
159	160	180	0
159	180	179	0
161	162	182	0
161	182	181	0
162	163	183	0
162	183	182	0
163	164	184	0
163	184	183	0
164	165	185	0
164	185	184	0
165	166	186	0
165	186	185	0
166	167	187	0
166	187	186	0
167	168	188	0
167	188	187	0
168	169	189	0
168	189	188	0
169	170	190	0
169	190	189	0
170	171	191	0
170	191	190	0
171	172	192	0
171	192	191	0
172	173	193	0
172	193	192	0
173	174	194	0
173	194	193	0
174	175	195	0
174	195	194	0
175	176	196	0
175	196	195	0
176	177	197	0
176	197	196	0
177	178	198	0
177	198	197	0
178	179	199	0
178	199	198	0
179	180	200	0
179	200	199	0
181	182	202	0
181	202	201	0
182	183	203	0
182	203	202	0
183	184	204	0
183	204	203	0
184	185	205	0
184	205	204	0
185	186	206	0

185	206	205	0
186	187	207	0
186	207	206	0
187	188	208	0
187	208	207	0
188	189	209	0
188	209	208	0
189	190	210	0
189	210	209	0
190	191	211	0
190	211	210	0
191	192	212	0
191	212	211	0
192	193	213	0
192	213	212	0
193	194	214	0
193	214	213	0
194	195	215	0
194	215	214	0
195	196	216	0
195	216	215	0
196	197	217	0
196	217	216	0
197	198	218	0
197	218	217	0
198	199	219	0
198	219	218	0
199	200	220	0
199	220	219	0
201	202	222	0
201	222	221	0
202	203	223	0
202	223	222	0
203	204	224	0
203	224	223	0
204	205	225	0
204	225	224	0
205	206	226	0
205	226	225	0
206	207	227	0
206	227	226	0
207	208	228	0
207	228	227	0
208	209	229	0
208	229	228	0
209	210	230	0
209	230	229	0
210	211	231	0
210	231	230	0
211	212	232	0
211	232	231	0
212	213	233	0
212	233	232	0
213	214	234	0
213	234	233	0
214	215	235	0
214	235	234	0
215	216	236	0
215	236	235	0
216	217	237	0
216	237	236	0
217	218	238	0
217	238	237	0
218	219	239	0
218	239	238	0
219	220	240	0
219	240	239	0
221	222	242	0
221	242	241	0
222	223	243	0
222	243	242	0

223	224	244	0
223	244	243	0
224	225	245	0
224	245	244	0
225	226	246	0
225	246	245	0
226	227	247	0
226	247	246	0
227	228	248	0
227	248	247	0
228	229	249	0
228	249	248	0
229	230	250	0
229	250	249	0
230	231	251	0
230	251	250	0
231	232	252	0
231	252	251	0
232	233	253	0
232	253	252	0
233	234	254	0
233	254	253	0
234	235	255	0
234	255	254	0
235	236	256	0
235	256	255	0
236	237	257	0
236	257	256	0
237	238	258	0
237	258	257	0
238	239	259	0
238	259	258	0
239	240	260	0
239	260	259	0
241	242	262	0
241	262	261	0
242	243	263	0
242	263	262	0
243	244	264	0
243	264	263	0
244	245	265	0
244	265	264	0
245	246	266	0
245	266	265	0
246	247	267	0
246	267	266	0
247	248	268	0
247	268	267	0
248	249	269	0
248	269	268	0
249	250	270	0
249	270	269	0
250	251	271	0
250	271	270	0
251	252	272	0
251	272	271	0
252	253	273	0
252	273	272	0
253	254	274	0
253	274	273	0
254	255	275	0
254	275	274	0
255	256	276	0
255	276	275	0
256	257	277	0
256	277	276	0
257	258	278	0
257	278	277	0
258	259	279	0
258	279	278	0
259	260	280	0

259	280	279	0
261	262	282	0
261	282	281	0
262	263	283	0
262	283	282	0
263	264	284	0
263	284	283	0
264	265	285	0
264	285	284	0
265	266	286	0
265	286	285	0
266	267	287	0
266	287	286	0
267	268	288	0
267	288	287	0
268	269	289	0
268	289	288	0
269	270	290	0
269	290	289	0
270	271	291	0
270	291	290	0
271	272	292	0
271	292	291	0
272	273	293	0
272	293	292	0
273	274	294	0
273	294	293	0
274	275	295	0
274	295	294	0
275	276	296	0
275	296	295	0
276	277	297	0
276	297	296	0
277	278	298	0
277	298	297	0
278	279	299	0
278	299	298	0
279	280	300	0
279	300	299	0
281	282	302	0
281	302	301	0
282	283	303	0
282	303	302	0
283	284	304	0
283	304	303	0
284	285	305	0
284	305	304	0
285	286	306	0
285	306	305	0
286	287	307	0
286	307	306	0
287	288	308	0
287	308	307	0
288	289	309	0
288	309	308	0
289	290	310	0
289	310	309	0
290	291	311	0
290	311	310	0
291	292	312	0
291	312	311	0
292	293	313	0
292	313	312	0
293	294	314	0
293	314	313	0
294	295	315	0
294	315	314	0
295	296	316	0
295	316	315	0
296	297	317	0
296	317	316	0

297	298	318	0
297	318	317	0
298	299	319	0
298	319	318	0
299	300	320	0
299	320	319	0
301	302	322	0
301	322	321	0
302	303	323	0
302	323	322	0
303	304	324	0
303	324	323	0
304	305	325	0
304	325	324	0
305	306	326	0
305	326	325	0
306	307	327	0
306	327	326	0
307	308	328	0
307	328	327	0
308	309	329	0
308	329	328	0
309	310	330	0
309	330	329	0
310	311	331	0
310	331	330	0
311	312	332	0
311	332	331	0
312	313	333	0
312	333	332	0
313	314	334	0
313	334	333	0
314	315	335	0
314	335	334	0
315	316	336	0
315	336	335	0
316	317	337	0
316	337	336	0
317	318	338	0
317	338	337	0
318	319	339	0
318	339	338	0
319	320	340	0
319	340	339	0
321	322	342	0
321	342	341	0
322	323	343	0
322	343	342	0
323	324	344	0
323	344	343	0
324	325	345	0
324	345	344	0
325	326	346	0
325	346	345	0
326	327	347	0
326	347	346	0
327	328	348	0
327	348	347	0
328	329	349	0
328	349	348	0
329	330	350	0
329	350	349	0
330	331	351	0
330	351	350	0
331	332	352	0
331	352	351	0
332	333	353	0
332	353	352	0
333	334	354	0
333	354	353	0
334	335	355	0

334	355	354	0
335	336	356	0
335	356	355	0
336	337	357	0
336	357	356	0
337	338	358	0
337	358	357	0
338	339	359	0
338	359	358	0
339	340	360	0
339	360	359	0
341	342	362	0
341	362	361	0
342	343	363	0
342	363	362	0
343	344	364	0
343	364	363	0
344	345	365	0
344	365	364	0
345	346	366	0
345	366	365	0
346	347	367	0
346	367	366	0
347	348	368	0
347	368	367	0
348	349	369	0
348	369	368	0
349	350	370	0
349	370	369	0
350	351	371	0
350	371	370	0
351	352	372	0
351	372	371	0
352	353	373	0
352	373	372	0
353	354	374	0
353	374	373	0
354	355	375	0
354	375	374	0
355	356	376	0
355	376	375	0
356	357	377	0
356	377	376	0
357	358	378	0
357	378	377	0
358	359	379	0
358	379	378	0
359	360	380	0
359	380	379	0
361	362	382	0
361	382	381	0
362	363	383	0
362	383	382	0
363	364	384	0
363	384	383	0
364	365	385	0
364	385	384	0
365	366	386	0
365	386	385	0
366	367	387	0
366	387	386	0
367	368	388	0
367	388	387	0
368	369	389	0
368	389	388	0
369	370	390	0
369	390	389	0
370	371	391	0
370	391	390	0
371	372	392	0
371	392	391	0

```

372      373      393      0
372      393      392      0
373      374      394      0
373      394      393      0
374      375      395      0
374      395      394      0
375      376      396      0
375      396      395      0
376      377      397      0
376      397      396      0
377      378      398      0
377      398      397      0
378      379      399      0
378      399      398      0
379      380      400      0
379      400      399      0
'N'      0 : (CONN1(:,K), REFT, K = 1,...,NELT1)
          : Printing option 'Y' or 'N'

Example 2
9          :1st geometry NLINES (m)
2.0000    2.0000    1.0000
-1.0000   -2.2361    0.0000
0.0000    0.0000    0.0000          : (COORCH(1,1:m))
-1.0000    1.0000    0.0000
0.0000    0.0000   -2.2361
-1.0000    1.0000    2.2361          : (COORCH(2,1:m))
10 1 2 0 1.0000 10 2 9 2 1.0000
10 9 5 2 1.0000 10 5 6 2 1.0000
10 6 1 2 1.0000 10 3 8 1 1.0000
10 8 4 1 1.0000 10 4 7 1 1.0000
10 7 3 1 1.0000          : (LINE(:,j),RATE(j),j=1,m)
2          :NCOMP (n, number of contours)
5          :number of lines in contour 1
1 2 3 4 5          :lines of contour 1
-4          :number of lines in contour 2
9 8 7 6          :lines of contour 2
4          :2nd geometry NLINES (m)
2.0000    6.0000    6.0000    2.0000          : (COORCH(1,1:m))
-1.0000   -1.0000    1.0000    1.0000          : (COORCH(2,1:m))
19 1 2 0 1.0000 10 2 3 0 1.0000
19 3 4 0 1.0000 10 4 1 0 1.0000 : (LINE(:,j),RATE(j),j=1,m)
1          :NCOMP (n, number of contours)
4          :number of lines in contour 1
1 2 3 4          :lines of contour 1
'N'          :Printing option 'Y' or 'N'

```

10.3 Program Results

nag_mesh2d_join (d06dbc) Example Program Results

Example 1

The restitched mesh characteristics
in the overlapping case

```

nv    = 785
nelt  = 1428
nedge = 152

```

in the partition case

```

nv    = 780
nelt  = 1444
nedge = 133

```

Example 2

(User-supplied callback fbnd, first invocation.)

The restitched mesh characteristics

```

nv    = 643
nelt  = 1133
nedge = 171

```

Example Program
Figure 1: Boundary and Interior Interface of Partitioned Squares

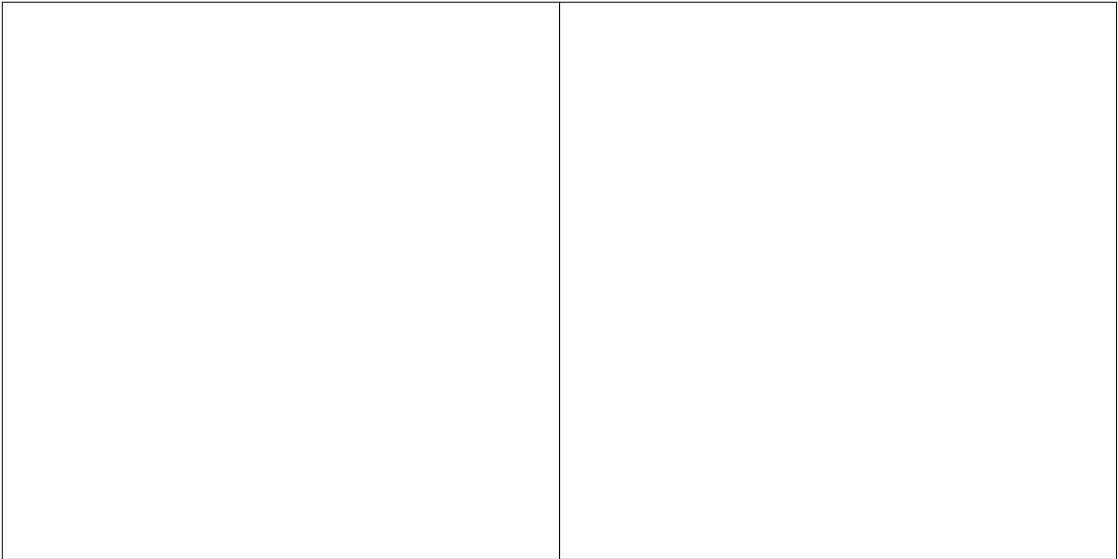


Figure 2: Interior Mesh of Partitioned Squares

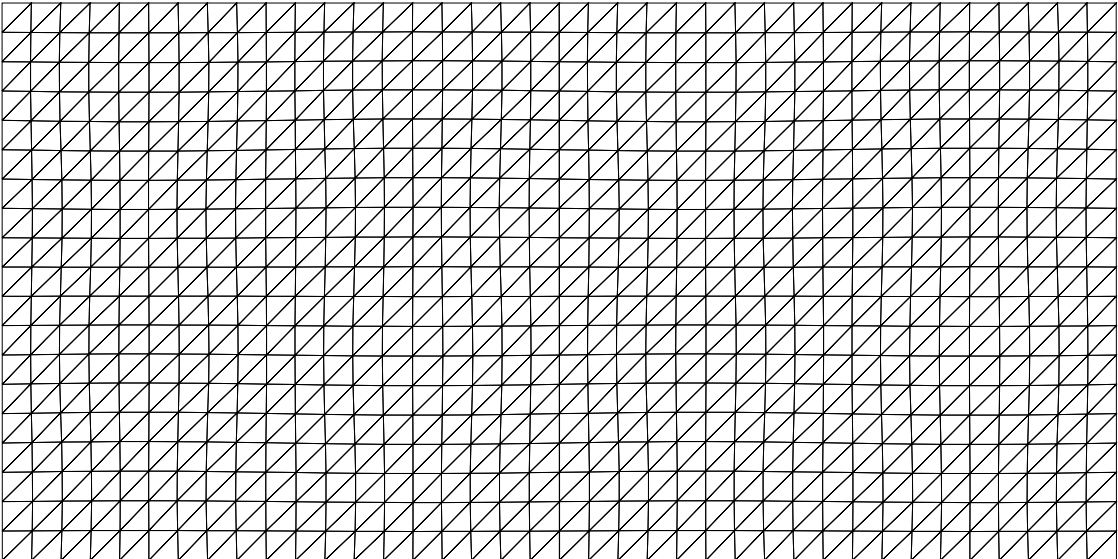


Figure 3: Boundary and Interior Interface of Overlapping Squares

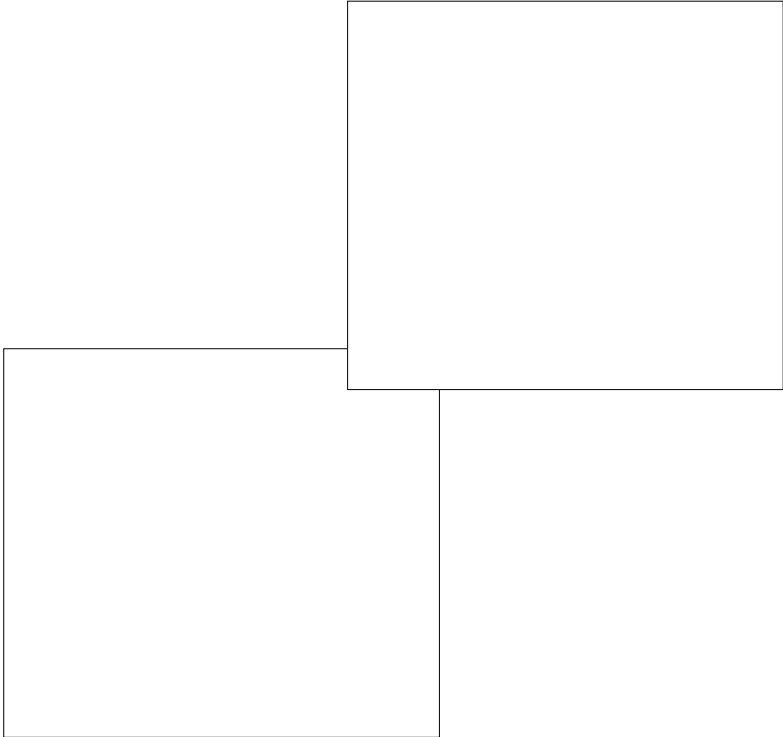


Figure 4: Interior Mesh of Overlapping Squares

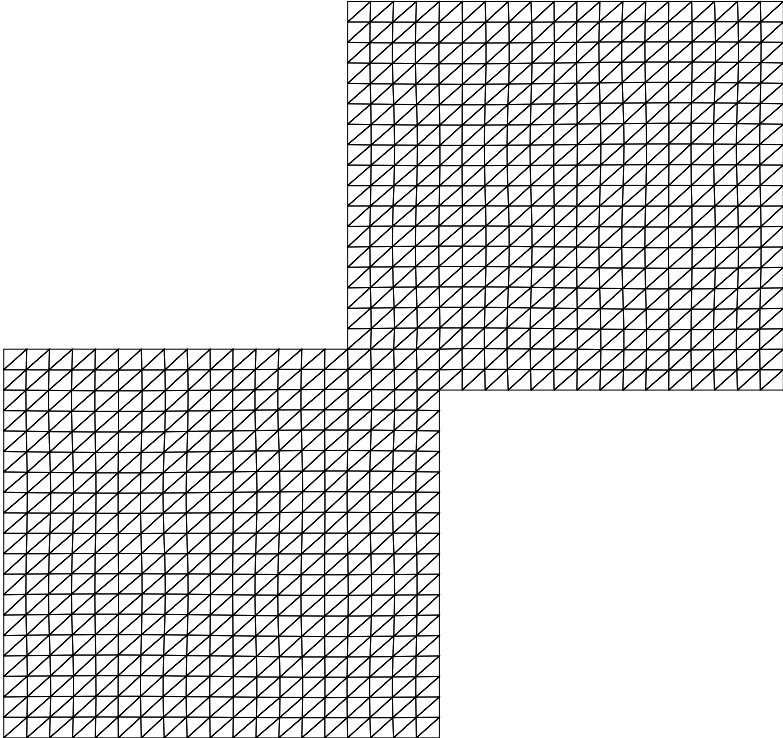


Figure 5: Boundary and Interior Interfaces for Key Shape

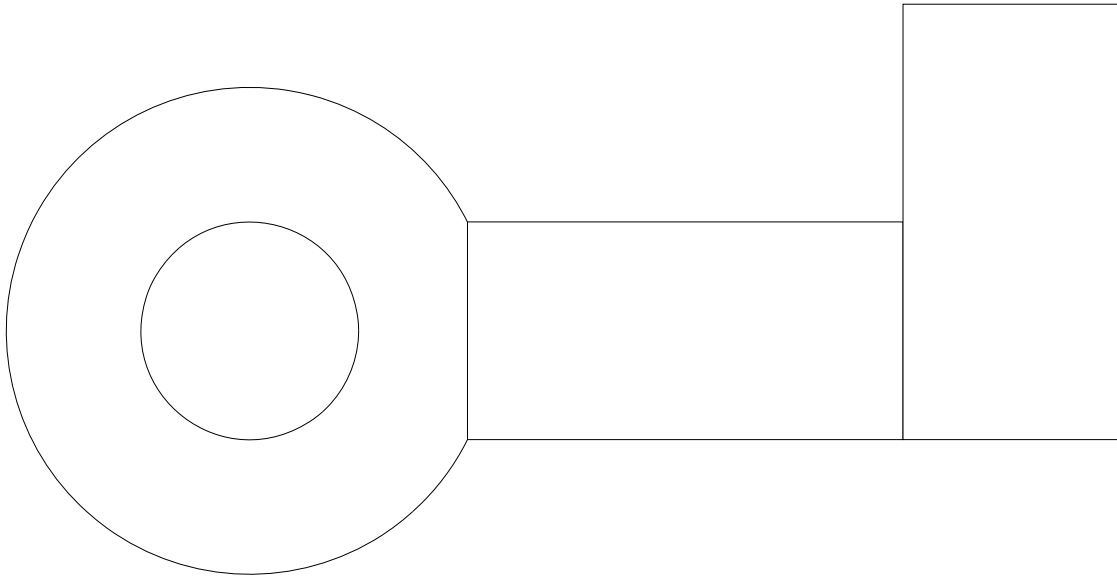


Figure 6: Interior Mesh of KeyShape

