NAG C Library Function Document

nag_mesh2d_trans (d06dac)

1 Purpose

nag_mesh2d_trans (d06dac) is a utility which performs an affine transformation of a given mesh.

2 Specification

```
void nag_mesh2d_trans (Integer mode, Integer nv, Integer nedge, Integer nelt,
      Integer ntrans, const Integer itype[], const double trans[], double coori[],
      Integer edgei[], Integer conni[], double cooro[], Integer edgeo[],
      Integer conno[], Integer itrace, const char *outfile, NagError *fail)
```

3 Description

nag_mesh2d_trans (d06dac) generates a mesh (coordinates, triangle/vertex connectivities and edge/vertex connectivities) resulting from an affine transformation of a given mesh. This transformation is of the form $y = A \times x + b$, where

y, x and b are in $\mathbb{R}^2$, and

A is a real 2 by 2 matrix.

Such a transformation includes a translation, a rotation, a scale reduction or increase, a symmetric transformation with respect to a user-supplied line, a user-supplied analytic transformation, or a composition of several transformations.

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 Parameters

1: **mode** – Integer *Input*

On entry: if $\mathbf{mode} = 1$ the arguments **coori**, **edgei** and **conni** are overwritten on exit by the output values described in **cooro**, **edgeo** and **conno** respectively. In this case **cooro**, **edgeo** and **conno** are not referenced, and the user can save storage space. If $\mathbf{mode} \neq 1$ no such aliasing is assumed.

2: **nv** – Integer *Input*

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

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

3: **nedge** – Integer *Input*

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

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

4: **nelt** – Integer *Input*

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

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

5: **ntrans** – Integer *Input*

On entry: the number of transformations of the input mesh.

Constraint: **ntrans** ≥ 1.

6: **itype**[**ntrans**] – const Integer *Input*

On entry: **itype**$[i-1]$, for $i = 1, \ldots,$ **ntrans**, indicates the type of each transformation as follows:

itype$[i-1] = 0$

Identity transformation.

itype$[i-1] = 1$

Translation.

itype$[i-1] = 2$

Symmetric transformation with respect to a user-supplied line.

itype$[i-1] = 3$

Rotation.

itype$[i-1] = 4$

Scaling.

itype$[i-1] = 10$

User-supplied analytic transformation.

Note that the transformations are applied in the order described in **itype**.

Constraint: **itype**$[i-1] = 0, 1, 2, 3, 4$ or 10 for $i = 1, 2, \ldots,$ **ntrans**.

7: **trans**[$6 \times$ **ntrans**] – const double *Input*

Note: where **TRANS**(i,j) appears in this document it refers to the array element **trans**$[6 \times (j-1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

On entry: the parameters for each transformation. For $i = 1, \ldots,$ **ntrans**, **TRANS**$(1,i)$ to **TRANS**$(6,i)$ contain the parameters of the ith transformation:

if **itype**$[i-1] = 0$, then elements **TRANS**$(1,i)$ to **TRANS**$(6,i)$ are not referenced;

if **itype**$[i-1] = 1$, then the translation vector is $\vec{u} = \begin{pmatrix} a \\ b \end{pmatrix}$, where $a =$ **TRANS**$(1,i)$ and $b =$ **TRANS**$(2,i)$, while elements **TRANS**$(3,i)$ to **TRANS**$(6,i)$ are not referenced;

if **itype**$[i-1] = 2$, then the user-supplied line is the curve $\{(x,y) \in \mathbb{R}^2;$ such that $ax + by + c = 0\}$, where $a =$ **TRANS**$(1,i)$, $b =$ **TRANS**$(2,i)$ and $c =$ **TRANS**$(3,i)$, while elements **TRANS**$(4,i)$ to **TRANS**$(6,i)$ are not referenced;

if **itype**$[i-1] = 3$, then the centre of the rotation is (x_0, y_0) where $x_0 =$ **TRANS**$(1,i)$ and $y_0 =$ **TRANS**$(2,i)$, $\theta =$ **TRANS**$(3,i)$ is its angle in degrees, while elements **TRANS**$(4,i)$ to **TRANS**$(6,i)$ are not referenced;

if **itype**$[i-1] = 4$, then $a =$ **TRANS**$(1,i)$ is the scaling coefficient in the x-direction, $b =$ **TRANS**$(2,i)$ is the scaling coefficient in the y-direction, and (x_0, y_0) are the scaling centre coordinates, with $x_0 =$ **TRANS**$(3,i)$ and $y_0 =$ **TRANS**$(4,i)$; while elements **TRANS**$(5,i)$ to **TRANS**$(6,i)$ are not referenced;

if **itype**$[i-1] = 10$, then the user-supplied analytic affine transformation $y = A \times x + b$ is such that $A = (a_{kl})_{1 \leq k,l \leq 2}$ and $b = (b_k)_{1 \leq k \leq 2}$ where $a_{kl} =$ **TRANS**$(2 \times (k-1) + l, i)$, and $b_k =$ **TRANS**$(4 + k, i)$ with $k, l = 1, 2$.

8: **coori**$[2 \times$ **nv**$]$ – double *Input/Output*

Note: where **COORI**(i, j) appears in this document it refers to the array element **coori**$[2 \times (j - 1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

On entry: **COORI**$(1, i)$ contains the x-coordinate of the ith vertex of the input mesh, for $i = 1, \ldots,$ **nv**; while **COORI**$(2, i)$ contains the corresponding y-coordinate.

On exit: if **mode** $= 1$, **coori** is assumed to hold the values of **cooro**.

9: **edgei**$[3 \times$ **nedge**$]$ – Integer *Input/Output*

Note: where **EDGEI**(i, j) appears in this document it refers to the array element **edgei**$[3 \times (j - 1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

On entry: the specification of the boundary or interface edges. **EDGEI**$(1, j)$ and **EDGEI**$(2, j)$ contain the vertex numbers of the two end-points of the jth boundary edge. **EDGEI**$(3, j)$ is a user-supplied tag for the jth boundary edge. Note that the edge vertices are numbered from 1 to **nv**.

On exit: if **mode** $= 1$, **edgei** holds the output values described in **edgeo**.

Constraint: $1 \leq$ **EDGEI**$(i, j) \leq$ **nv** and **EDGEI**$(1, j) \neq$ **EDGEI**$(2, j)$ for $i = 1, 2$ and $j = 1, 2, \ldots,$ **nedge**.

10: **conni**$[3 \times$ **nelt**$]$ – Integer *Input/Output*

Note: where **CONNI**(i, j) appears in this document it refers to the array element **conni**$[3 \times (j - 1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

On entry: the connectivity of the input mesh between triangles and vertices. For each triangle j, **CONNI**(i, j) 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**.

On exit: if **mode** $= 1$, **conni** holds the ouptut values described in **conno**.

Constraints:

$1 \leq$ **CONNI**$(i, j) \leq$ **nv**;
CONNI$(1, j) \neq$ **CONNI**$(2, j)$;
CONNI$(1, j) \neq$ **CONNI**$(3, j)$ and **CONNI**$(2, j) \neq$ **CONNI**$(3, j)$ for $i = 1, 2, 3$ and $j = 1, 2, \ldots,$ **nelt**.

11: **cooro**$[dim]$ – double *Output*

Note: where **COORO**(i, j) appears in this document it refers to the array element **cooro**$[2 \times (j - 1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

The dimension, dim, of the array **cooro** must be at least **nv** when **mode** $\neq 1$ and at least 1 otherwise.

On exit: **COORO**$(1, i)$ will contain the x-coordinate of the ith vertex of the transformed mesh, for $i = 1, \ldots,$ **nv**; while **COORO**$(2, i)$ will contain the corresponding y-coordinate. If **mode** $= 1$ the results are instead overwritten in **coori**.

12: **edgeo**$[dim]$ – Integer *Output*

Note: where **EDGEO**(i, j) appears in this document it refers to the array element **edgeo**$[3 \times (j - 1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

The dimension, dim, of the array **edgeo** must be at least **nedge** when **mode** $\neq 1$ and at least 1 otherwise.

On exit: the specification of the boundary or interface edges of the transformed mesh. If the number of symmetric transformations is even or zero then $\mathbf{EDGEO}(i,j) = \mathbf{EDGEI}(i,j)$ for $i = 1,2,3$ and $j = 1,\ldots,$ **nedge**; otherwise $\mathbf{EDGEO}(1,j) = \mathbf{EDGEI}(2,j)$, $\mathbf{EDGEO}(2,j) = \mathbf{EDGEI}(1,j)$ and $\mathbf{EDGEO}(3,j) = \mathbf{EDGEI}(3,j)$ for $j = 1,\ldots,$ **nedge**. If **mode** $= 1$ the results are overwritten in **edgei**.

13: **conno**$[dim]$ – Integer *Output*

Note: where $\mathbf{CONNO}(i,j)$ appears in this document it refers to the array element **conno**$[3 \times (j-1) + i - 1]$. We recommend using a #define to make the same definition in your calling program.

The dimension, dim, of the array **conno** must be at least **nelt** when **mode** $\neq 1$ and at least 1 otherwise.

On exit: the connectivity of the transformed mesh between triangles and vertices. If the number of symmetric transformations is even or zero then $\mathbf{CONNO}(i,j) = \mathbf{CONNI}(i,j)$ for $i = 1,2,3$ and $j = 1,\ldots,$ **nelt**; otherwise $\mathbf{CONNO}(1,j) = \mathbf{CONNI}(1,j)$, $\mathbf{CONNO}(2,j) = \mathbf{CONNI}(3,j)$ and $\mathbf{CONNO}(3,j) = \mathbf{CONNI}(2,j)$, for $j = 1,\ldots,$ **nelt**. Note that the mesh vertices are numbered from 1 to **nv**. If **mode** $= 1$ the results are instead overwritten in **conni**.

14: **itrace** – Integer *Input*

On entry: the level of trace information required from nag_mesh2d_trans (d06dac) as follows:

if **itrace** ≤ 0, no output is generated;

if **itrace** ≥ 1, then details of each transformation, the matrix A and the vector b of the final transformation, which is the composition of all the **ntrans** transformations, are printed.

15: **outfile** – 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.

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

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

NE_INT

On entry, **ntrans** $= \langle$*value*$\rangle$
Constraint: **ntrans** > 0.

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

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

NE_INT_2

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

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

On entry, **itype**$[i-1]$ is not equal to 0, 1, 2, 3, 4, 10, **itype**$[i-1] = \langle$*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*$\rangle$.

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

NE_INT_4

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

On entry, **conni**$(i,j) < 1$ or **conni**$(i,j) >$ **nv**, where **conni**(i,j) denotes **conni**$[3 \times (j-1) + i - 1]$: **conni**$(i,j) = \langle$*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 routine. Check the input mesh especially the connectivities and the details of each transformations.

NE_ALLOC_FAIL

Memory allocation failed.

NE_BAD_PARAM

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

NE_NOT_WRITE_FILE

Cannot open file $\langle$*value*$\rangle$ for writing.

NE_NOT_CLOSE_FILE

Cannot close file $\langle$*value*$\rangle$.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

For an example of the use of this utility function, see nag_mesh2d_join (d06dbc).