Nonlinear Shell MITC4

Constructor for Stiffness and Internal Force:

```
NonlinearShellMITC4(vector<vector<double>> &xyz_in,
                    vector<vector<double>> &vnor_in,
                    vector<double> &thick_in,
                    vector<vector<double>> &xlocal_in,
                    vector<vector<double>> &ylocal_in,
                    vector<double> &zetGPcoord_in,
                    vector<double> &zetGPweigth_in,
                    vector<double> &stresses_in,
                    vector<double> &DMatrix_in);
```

Required (not null or empty) inputs:

xyz_in = nodal coordinates (e.g. xyz_in[2][0] is the x-coordinate of the third node of the element, xyz_in[0][1] is the y-coordinate of the first node of the element). Example of this variable's definition is given on the free template distributed with the API.

vnor_in = nodal normal vector components (e.g. vnor_in[2][0] is the normal vector's x-component of the third node of the element, xyz_in[0][1] is the normal vector's y-component of the first node of the element). Example of this variable's definition is given on the free template distributed with the API.

thick_in = thickness at the nodes of the element (e.g. thick_in[3] is the thickness value for the fourth node of the element). Example of this variable's definition is given on the free template distributed with the API.

zetGPcoord_in = through the thickness Gauss integration coordinates defined in the natural coordinate system. A minimum of 2 integration points must be used. Example of this variable's definition for a 2-points Gauss integration scheme is given on the free template distributed with the API.

zetGPweigth_in = through the thickness Gauss integration weights. A minimum of 2 integration points must be used. Example of this variable's definition for a 2-points Gauss integration scheme is given on the free template distributed with the API.

stresses_in = stresses at the Gauss integration points of the element. This input is required as "non-empty" for the calculation of the internal force vector and the geometric (initial stress) stiffness matrix for nonlinear analysis.

DMatrix_in = constitutive matrix in vector form for all integration points. Each integration point has 25 (5x5) positions in the D-Matrix vector. This vector will have to have the following definition before passed in the constructor:

```
for (int izet = 0; izet < zetCoord.size(); ++izet) {
	for (int ig = 0; ig < 4; ++ig) {
		for (int i = 0; i < 25; ++i) {
			DMatrix_in.push_back(DMatrix[i]);
		}
	}
}
```

where “Dmatrix” has the 5x5 constitutive matrix in vector form. This construction allows for the constitutive matrix to be defined for linear elastic (Hooke’s law) analysis or for the elasto-plastic tangent constitutive matrix for plasticity. Example of this variable’s definition is given on the free template distributed with the API for a linear elastic example.

Optional (empty) inputs:

xlocal_in and ylocal_in = nodal local coordinate axes at the nodes. Together with “vnor_in” they form a local nodal coordinate system. The API constructs these vectors automatically if both “xlocal_in” and “ylocal_in” are passed through as empty vectors. Example of this variable’s definition is given on the free template distributed with the API.

stresses_in = stresses at the Gauss integration points of the element. For linear analysis, this input can be passed as an empty variable because it is not required for the construction of the tangent linear elastic stiffness matrix. Example of this variable’s definition for the calculation of the linear elastic stiffness matrix is given on the free template distributed with the API.
The format of the stresses input is the following:
Sxx, Syy, Sxy, Sxz, Syz, that is, 5 stress components per integration point
The order of the integration points in the "stresses_in" vector is from the bottom-up direction, from zet -1.0 to +1.0. For example, for 2 integration points along thickness, zet1 = -1.0/sqrt(3.0) and zet2 = +1.0/sqrt(3.0), we get: stresses_in[Sxx_I^zet1, Syy_I^zet1, Sxy_I^zet1, Sxz_I^zet1, Syz_I^zet1,..., Sxx_IV^zet1, Syy_IV^zet1, Sxy_IV^zet1, Sxz_IV^zet1, Syz_IV^zet1,Sxx_I^zet2, Syy_I^zet2, Sxy_I^zet2, Sxz_I^zet2, Syz_I^zet2,..., Sxx_IV^zet2, Syy_IV^zet2, Sxy_IV^zet2, Sxz_IV^zet2, Syz_IV^zet2].

Constructor for the Strains:

```
NonlinearShellMITC4(vector<vector<double>> &xyz_in,
                    vector<vector<double>> &vnor_in,
                    vector<double> &thick_in,
                    vector<vector<double>> &xlocal_in,
                    vector<vector<double>> &ylocal_in,
                    vector<double> &zetGPcoord_in,
                    vector<double> &Displacements_in);
```

Required (not null or empty) inputs:

xyz_in = nodal coordinates (e.g. xyz_in[2][0] is the x-coordinate of the third node of the element, xyz_in[0][1] is the y-coordinate of the first node of the element). Example of this variable’s definition is given on the free template distributed with the API.

vnor_in = nodal normal vector components (e.g. vnor_in[2][0] is the normal vector's x-component of the third node of the element, xyz_in[0][1] is the normal vector's y-component of the first node of the element). Example of this variable's definition is given on the free template distributed with the API.

thick_in = thickness at the nodes of the element (e.g. thick_in[3] is the thickness value for the fourth node of the element). Example of this variable's definition is given on the free template distributed with the API.

zetGPcoord_in = through the thickness Gauss integration coordinates defined in the natural coordinate system. A minimum of 2 integration points must be used. Example of this variable's definition for a 2-points Gauss integration scheme is given on the free template distributed with the API.

Displacements_in = nodal displacement vector components (e.g. Displacements_in[2] is the z-displacement of node 1, Displacements_in[3] is the rotation of the xlocal_in axis of node 1, Displacements_in[4] is the rotation of the ylocal_in axis of node 1, and Displacements_in[5] is always 0.0. Displacements_in[12] is the x-displacement of node 3, Displacements_in[19] is the y-displacement of node 4). Example of this variable's definition is given on the free template distributed with the API.

Optional (empty) inputs:

xlocal_in and ylocal_in = nodal local coordinate axes at the nodes. Together with "vnor_in" they form a local nodal coordinate system. The API constructs these vectors automatically if both "xlocal_in" and "ylocal_in" are passed through as empty vectors. Example of this variable's definition is given on the free template distributed with the API.

Public Methods (you have access to):

```
void get_BMatrix(int igaus_in, double zet_in, vector<double> *BMat_out);
```
gets the strain-displacement "B-Matrix" for the in-plane Gauss point "igaus_in" (in-plane Gauss points are fixed and equal to 4 GPs) and for the GP along the thickness direction as defined by the input variable "zet_in". The "B-Matrix" is in vector form (vector<double>).

```
void get_BMatrix_Transposed(vector<double> *BMatT) { *BMatT = BMatTransposed_; }
```
Transposed of the "B-Matrix". The "get_BMatrix" method must be called first to construct the B-Matrix and only then the transposed B-Matrix can be called. This method is helpful because of the vector form in which the "B-Matrix" is stored.

```
void set_LocalAxes_All(vector<double> axesgp_in) { axesgp_all_ = axesgp_in; }
```
defines the local axes at the in-plane Gauss points of the element. The API automatically calculates these axes but the user might want to define the axes for example for modelling of planar plastic anisotropy or for composites. In these cases, these axes should be defined after the Constructor. The format of the GP axes array (axesgp_in) must be the following:

```
for (int ig = 0; ig < 4; ++ig) {
        for (int i = 0; i < 9; ++i) {
                axesgp_in.push_back(axes[i]);
        }
}
```

with “axes” a vector of size 9 (x-local vector, followed by y-local vector and z-local vector). Please note that the local axes at the GPs are defined for zet = 0.0, i.e. at the mid-surface of the element.

`vector<double> get_LocalAxes_All() { return axesgp_all_; }`
get the local axes at all Gauss integration points.

`vector<double> Calculate_StiffnessMITC4();`
calculates the linear tangent stiffness matrix of the element. The stiffness matrix for this element is a 24x24 matrix but it is stored in vector form for better computational efficiency. The first row of 24 stiffness values are first elements in the vector, followed by the second row 24 of elements and so on.

`vector<double> get_StiffnessMITC4() { return Stiffness_; }`
getter for the linear tangent stiffness matrix of the element. It return the stiffness in vector form (vector<double>). It should be called only after the calculation of the stiffness using the method “Calculate_StiffnessMITC4()” as described above.

`vector<double> get_Stiffness_InitialStress();`
gets the geometric stiffness matrix (or the initial stress stiffness matrix) of the element. The geometric stiffness matrix for this element is a 24x24 matrix but it is stored in vector form for better computational efficiency. The first row of 24 stiffness values are the first elements in the vector, followed by the second row 24 of elements and so on.

`vector<double> Calculate_Internal_Forces();`
calculates the internal force vector for the element.

`vector<double> get_Internal_Forces() { return Internal_Force_; }`
gets the internal force vector for the element. It must be called only after the calculation of the internal force vector of the element at “Calculate_Internal_Forces()”.

`void Calculate_Stiffness_Internal();`
calculates both the linear tangent stiffness and internal force vector. Ideal for nonlinear analysis for saving computational time.

`vector<double> get_StrainsMITC4_Global();`
gets the strain tensor at the global coordinate system.
The format of the strains is the following:
Exx, Eyy, Exy, Exz, Eyz, that is, 5 stress components per integration point.
The order of the integration points is from the bottom-up direction, from zet -1.0 to +1.0. For example, for 2 integration points along thickness, zet1 = -1.0/sqrt(3.0) and zet2 = +1.0/sqrt(3.0), we get: [Exx_I^zet1, Eyy_I^zet1, Exy_I^zet1, Exz_I^zet1, Eyz_I^zet1,..., Exx_IV^zet1, Eyy_IV^zet1, Exy_IV^zet1, Exz_IV^zet1, Eyz_IV^zet1, Exx_I^zet2, Eyy_I^zet2, Exy_I^zet2, Exz_I^zet2, Eyz_I^zet2, … , Exx_IV^zet2, Eyy_IV^zet2, Exy_IV^zet2, Exz_IV^zet2, Eyz_IV^zet2].

`void rotate_GP_axes();`
rotates the local axes at the Gauss points of the element. This is required for nonlinear analysis.