My Project

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 NonlinearShellMITC4 Class Reference

Public Member Functions

- NonlinearShellMITC4 (string path_in, 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)

    Constructor for a new Nonlinear Shell MITC4 object. This constructor can be used for the stiffness matrix and for the internal force vector.
    The procedure for calculculating the initial local nodal coordinate system (V1, V2, V3) is the following:

- NonlinearShellMITC4 (string path_in, 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, vector< double > &DMatrix_in)

    Constructor for a new Nonlinear Shell MITC4 object. This constructor can be used for the strains and stresses.
    The procedure for calculculating the initial local nodal coordinate system (V1, V2, V3) is the following:

- size_t **get_Rows** ()
- size_t **get_Cols** ()
- void get_BMatrix (int igaus_in, int izet_in, double zet_in, vector< double > *BMat_out)

    *Calculates the Strain-Displacement matrix (B-matrix) at the Gauss point defined by "izet_in" and "igaus_in". *** Strains = B-matrix . displ ***.*

- void get_BMatrix_Transposed (vector< double > *BMatT)

    Get the Strain-Displacement matrix transposed (B-matrix)^ T at the Gauss point defined by "izet_in" and "igaus_in". warning: the method "get_BMatrix" must be called first to construct the B-Matrix at the corresponding Gauss point. Only then, this method can be called for the transposed Strain-Displacement matrix.

- void set_LocalAxes_All (vector< double > axesgp_in)

    Set the local axes at the Gauss integration points of the element. If the local axes are not setted then the API will calculate the local axes at the GPs. If the MITC4 shell is to be used for composites or planar plastic anisotropy then the local axes at the GPs should be setted after the instantiation of the shell MITC4 class.

- vector< double > get_LocalAxes_All ()

    Get the local axes at all integration points of the shell element. The return is a vector with the axes at all GPs which is stored in the same order as explained in the setter - "set_LocalAxes_All".

- void **Calculate_Local_Axes_All_GP** ()

    calculates the local axes at each in-plane Gauss point (located at the mid-surface (zet = 0.0) of the shell). The axes at zet = 0.0 are then copied to all other Gauss points of the MITC4 shell element.

- vector< double > Calculate_StiffnessMITC4 ()

  Calculates the linear stiffness matrix "Stiffness_" of the MITC4 shell element. The calculation of the stiffness matrix was left on a separated method because we might want to instantiate this class for other features than the stiffness matrix and therefore we don't need to waste CPU time of calculating the stiffness matrix if it is not needed.
- vector< double > get_StiffnessMITC4 ()

  Get the stiffness matrix "Stiffness_" in vector form. This is a getter only for the stiffness matrix, it does not calculate the stiffness matrix.
- vector< double > get_Stiffness_InitialStress ()

  Calculates the Initial Stress Stiffness matrix (or geometric stiffness) for the MITC4 shell element. This stiffness is fundamental for nonlinear continuum analysis.
- vector< double > Calculate_Internal_Forces ()

  Calculates the Internal Force vector "Internal_Force_" for the MITC4 shell element. The stresses must be passed first to the instance of this element through its constructor. Note that the right constructor must be used.
- vector< double > get_Internal_Forces ()

  Getter for the Internal Force vector "Internal_Force_".
- void **Calculate_Stiffness_Internal** ()

  Compute the total stiffness matrix and the internal force vector. The total stiffness is the sum of the linear stiffness (Stiffness_) and the initial-stress stiffness (StiffnessInitialStress_).
  From a CPU-efficiency perspective, in nonlinear continuum mechanics it is advantageous to compute the stiffness matrix and the internal force vector simultaneously.
  After calling this method, the method "get_StiffnessMITC4()" must be called for getting the total stiffness, which is stored in the vector "Stiffness_", and the method "get_Internal_Forces()" for getting the internal force vector.
- vector< double > get_StrainsMITC4 ()

  Get the StrainsMITC4 object.
- vector< double > **get_StrainsMITC4_Global** ()
- void **rotate_GP_axes** ()
- vector< double > **get_StressesMITC4** ()
- vector< double > **get_StressesMITC4_Global** ()
- void **Jacobi** (vector< vector< double > > &ain, vector< vector< double > > &bin, vector< double > &eigenvalues_, vector< vector< double > > &eigenvectors_)

Static Public Attributes

- static string **path** = ""

3.1.1 Constructor & Destructor Documentation

3.1.1.1 NonlinearShellMITC4() [1/2]

```
NonlinearShellMITC4::NonlinearShellMITC4 (
            string path_in,
            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 )
```

Constructor for a new Nonlinear Shell MITC4 object. This constructor can be used for the stiffness matrix and for the internal force vector.
The procedure for calculculating the initial local nodal coordinate system (V1, V2, V3) is the following:

1. define a vector: vec1 = {0.0, 1.0, 0.0};
2. define "xloc" as the result of the following cross-product: xloc = vec1 x V3 (V3 is the normal vector at the node);
3. if (xloc.empty()) { xloc = {1.0, 0.0, 0.0} };
4. V1 = xloc;
5. V2 = V3 x V1.

Parameters

path_in	String with the full path to the license file.
xyz_in	Nodal coordinates of the four shell element nodes, stored in xyz_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z coordinates.
vnor_in	Normal vector of the four shell element nodes, stored in vnor_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the normal vector.
thick_in	Element's thickness at the nodes, stored in thick_in[0:3], where each index is a node of the MITC4 shell element.
xlocal_in	V1 local nodal vector stored in xlocal_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the "xlocal_in" vector. If passed empty then it will be built inside the constructor. V1 (xlocal_in),V2 (ylocal_in) and V3 (vnor_in) make a local nodal coordinate system at each node.
ylocal_in	V2 local nodal vector stored in ylocal_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the "ylocal_in" vector. If passed empty then it will be built inside the constructor. V1 (xlocal_in),V2 (ylocal_in) and V3 (vnor_in) make a local nodal coordinate system at each node.
zetGPcoord_in	Natural coordinates of the Gauss integration points along the thickness direction of the shell. The number of integration points along the thickness direction is defined here through the size of this vector.
zetGPweigth↩_in	Weights of the Gauss integration points along the thickness direction of the shell. The size of this vector must be the same as the size of "zetGPcoord_in".
stresses_in	The stresses at the integration points must be provided as input to the class constructor. These stresses are required for the computation of the internal force vector and the initial stress stiffness matrix. The stress input format is defined as follows: Sxx, Syy, Sxy, Sxz, Syz, i.e., five stress components per integration point. The stresses are stored in the "stresses_" vector, ordered by integration points through the thickness in a bottom-to-top direction, from zet = -1.0 to zet = +1.0. For example, with two integration points through the thickness, zet1 = -1.0 / sqrt(3.0) and zet2 = +1.0 / sqrt(3.0), the ordering in stresses_ is: [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]

Parameters

DMatrix_in	Plane stress constitutive matrix, stored in vector form with size = 25. The "DMatrix_in" vector must contain the 5x5 constitutive matrix for all integration points of the shell element. The procedure to construct this vector is the following:
	1. Loop in the integration points along thickness direction: for (int izet = 0; izet < zetGPCoord_in.size(); ++izet) {
	2. Loop in the 4 in-plane GPs: for (int ig = 0; ig < 4; ++ig) {
	3. Insert the 5x5 D-matrix, stored in vector form ("DMatrix[0:24]"), into the "DMatrix_in" vector:
	3.1 for (int i = 0; i < 25; ++i) { DMatrix_all.push_back(DMatrix[i]); }

3.1.1.2 NonlinearShellMITC4() [2/2]

```
NonlinearShellMITC4::NonlinearShellMITC4 (
            string path_in,
            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,
            vector< double > & DMatrix_in )
```

Constructor for a new Nonlinear Shell MITC4 object. This constructor can be used for the strains and stresses. The procedure for calculculating the initial local nodal coordinate system (V1, V2, V3) is the following:

1. define a vector: vec1 = {0.0, 1.0, 0.0};
2. define "xloc" as the result of the following cross-product: xloc = vec1 x V3 (V3 is the normal vector at the node);
3. if (xloc.empty()) { xloc = {1.0, 0.0, 0.0} };
4. V1 = xloc;
5. V2 = V3 x V1.

Parameters

path_in	String with the full path to the license file.
xyz_in	Nodal coordinates of the four shell element nodes, stored in xyz_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z coordinates.

Parameters

vnor_in	Normal vector of the four shell element nodes, stored in vnor_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the normal vector.
thick_in	Element's thickness at the nodes, stored in thick_in[0:3], where each index is a node of the MITC4 shell element.
xlocal_in	V1 local nodal vector stored in xlocal_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the "xlocal_in" vector. If passed empty then it will be built inside the constructor. V1 (xlocal_in),V2 (ylocal_in) and V3 (vnor_in) make a local nodal coordinate system at each node.
ylocal_in	V2 local nodal vector stored in ylocal_in[0:3][0:2], where each row corresponds to a node of the element and each column corresponds to the x,y, and z components of the "ylocal_in" vector. If passed empty then it will be built inside the constructor. V1 (xlocal_in),V2 (ylocal_in) and V3 (vnor_in) make a local nodal coordinate system at each node.
zetGPcoord_in	Natural coordinates of the Gauss integration points along the thickness direction of the shell. The number of integration points along the thickness direction is defined here through the size of this vector.
Displacements_in	Displacement vector for the element. The shell MITC4 has 4 nodes and each node has 6 degrees of freedom. The size of this vector is 4x6 = 24.

DMatrix_in	Plane stress constitutive matrix, stored in vector form with size = 25. The "DMatrix_in" vector must contain the 5x5 constitutive matrix for all integration points of the shell element. The procedure to construct this vector is the following:
	1. Loop in the integration points along thickness direction: for (int izet = 0; izet < zetGPCoord_in.size(); ++izet) {
	2. Loop in the 4 in-plane GPs: for (int ig = 0; ig < 4; ++ig) {
	3. Insert the 5x5 D-matrix, stored in vector form ("DMatrix[0:24]"), into the "DMatrix_in" vector:
	3.1 for (int i = 0; i < 25; ++i) { DMatrix_all.push_back(DMatrix[i]); }

3.1.2 Member Function Documentation

3.1.2.1 Calculate_Internal_Forces()

```
vector< double > NonlinearShellMITC4::Calculate_Internal_Forces ( )
```

Calculates the Internal Force vector "Internal_Force_" for the MITC4 shell element. The stresses must be passed first to the instance of this element through its constructor. Note that the right constructor must be used.

Returns

vector<double> The return is the Internal Force vector of size = 24.

3.1.2.2 Calculate_StiffnessMITC4()

```
vector< double > NonlinearShellMITC4::Calculate_StiffnessMITC4 ( )
```

Calculates the linear stiffness matrix "Stiffness_" of the MITC4 shell element. The calculation of the stiffness matrix was left on a separated method because we might want to instantiate this class for other features than the stiffness matrix and therefore we don't need to waste CPU time of calculating the stiffness matrix if it is not needed.

Returns

vector<double> The return is the stiffness matrix "Stiffness_" in vector form (of size = 576). The first elements in the output vector (Stiffness_[0:23]) correspond to the first 24 elements in the first row of the 24 x 24 stiffness (stif[0][0:23]), the next 24 elements in the output vector (Stiffness_[24:47]) correspond to stif[1][0:23], and so on.

3.1.2.3 get_BMatrix()

```
void NonlinearShellMITC4::get_BMatrix (
            int igaus_in,
            int izet_in,
            double zet_in,
            vector< double > * BMat_out )
```

Calculates the Strain-Displacement matrix (B-matrix) at the Gauss point defined by "izet_in" and "igaus_in". *** Strains = B-matrix . displ ***.

Parameters

igaus_in	Number of the in-plane Gauss point (0:3).
izet_in	Number of the integration point along the thickness direction.
zet_in	Natural coordinate of the integration point along the thickness direction.
BMat_out	Strain-Displacement matrix of the MITC4 shell element in vector form and defined in the local GP coordinate system (output). This vector is organised as follows: BMat_out[0:23] corresponds to "Strains_xx"; BMat_out[24:47] corresponds to "Strains_yy"; BMat_out[48:71] corresponds to "Strains_xy"; BMat_out[72:95] corresponds to "Strains_xz"; BMat_out[96:119] corresponds to "Strains_yz";

3.1.2.4 get_BMatrix_Transposed()

```
void NonlinearShellMITC4::get_BMatrix_Transposed (
            vector< double > * BMatT )  [inline]
```

Get the Strain-Displacement matrix transposed (B-matrix)^T at the Gauss point defined by "izet_in" and "igaus_in". warning: the method "get_BMatrix" must be called first to construct the B-Matrix at the corresponding Gauss point. Only then, this method can be called for the transposed Strain-Displacement matrix.

Parameters

BMatT	Transposed Strain-Displacement matrix stored in vector form (output).

3.1.2.5 get_Internal_Forces()

`vector< double > NonlinearShellMITC4::get_Internal_Forces ( ) [inline]`

Getter for the Internal Force vector "Internal_Force_".

Returns

vector<double> The return is the Internal Force vector "Internal_Force_" of size = 24.

3.1.2.6 get_LocalAxes_All()

`vector< double > NonlinearShellMITC4::get_LocalAxes_All ( ) [inline]`

Get the local axes at all integration points of the shell element. The return is a vector with the axes at all GPs which is stored in the same order as explained in the setter - "set_LocalAxes_All".

Returns

vector<double>

3.1.2.7 get_Stiffness_InitialStress()

`vector< double > NonlinearShellMITC4::get_Stiffness_InitialStress ( )`

Calculates the Initial Stress Stiffness matrix (or geometric stiffness) for the MITC4 shell element. This stiffness is fundamental for nonlinear continuum analysis.

Returns

vector<double> The return is the Initial Stress Stiffness matrix, "StiffnessInitialStress_", stored in vector form (of size = 576). The first elements in the output vector (StiffnessInitialStress_[0:23]) correspond to the first 24 elements in the first row of the 24 x 24 "StifInitialStress" (StifInitialStress[0][0:23]), the next 24 elements in the output vector (StiffnessInitialStress_[24:47]) correspond to StifInitialStress[1][0:23], and so on.

3.1.2.8 get_StiffnessMITC4()

`vector< double > NonlinearShellMITC4::get_StiffnessMITC4 ( ) [inline]`

Get the stiffness matrix "Stiffness_" in vector form. This is a getter only for the stiffness matrix, it does not calculate the stiffness matrix.

Returns

vector<double> The return is the stiffness matrix "Stiffness_" in vector form (of size = 576). The first elements in the output vector (Stiffness_[0:23]) correspond to the first 24 elements in the first row of the 24 x 24 stiffness (stif[0][0:23]), the next 24 elements in the output vector (Stiffness_[24:47]) correspond to stif[1][0:23], and so on.

3.1.2.9 get_StrainsMITC4()

```
vector< double > NonlinearShellMITC4::get_StrainsMITC4 ( )
```

Get the StrainsMITC4 object.

Returns

vector<double>

3.1.2.10 set_LocalAxes_All()

```
void NonlinearShellMITC4::set_LocalAxes_All (
            vector< double > axesgp_in )  [inline]
```

Set the local axes at the Gauss integration points of the element. If the local axes are not setted then the API will calculate the local axes at the GPs. If the MITC4 shell is to be used for composites or planar plastic anisotropy then the local axes at the GPs should be setted after the instantiation of the shell MITC4 class.

Parameters

axesgp_in	the structure for "axesgp_in" must be the following:
	Loop in the integration points along thickness direction: for (int izet = 0; izet < zetGPCoord_in.size(); ++izet) {
	Loop in the 4 in-plane GPs: for (int ig = 0; ig < 4; ++ig) {
	axesgp[0:2] = x-local vector
	axesgp[3:5] = y-local vector
	axesgp[6:8] = z-local vector
	for (int i = 0; i < axesgp.size(); ++i) { axesgp_in.push_back(axesgp[i]); } } }

The documentation for this class was generated from the following file:

- /home/mestrrc2/Documents/MechinMotionLtd/APIs/NonlinearShellMITC4/NonlinearShellMITC4.h

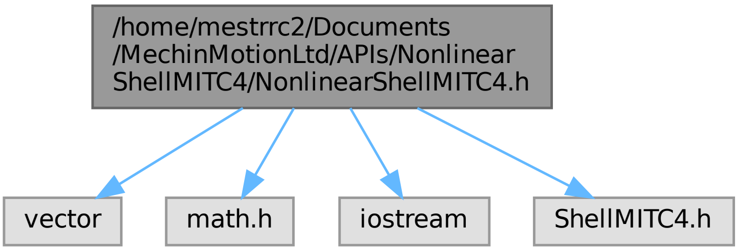
Chapter 4

File Documentation

4.1 /home/mestrrc2/Documents/MechinMotionLtd/APIs/NonlinearShell↩ MITC4/NonlinearShellMITC4.h File Reference

```
#include <vector>
#include <math.h>
#include <iostream>
#include "ShellMITC4.h"
```

Include dependency graph for NonlinearShellMITC4.h:



Classes

- class NonlinearShellMITC4

4.1.1 Detailed Description

Author

Rui Cardoso (support@mechinmotion.com)

Version

1.0

Date

2025-12-30

Copyright

4.2 NonlinearShellMITC4.h

Go to the documentation of this file.

```
#include <vector>
#include <math.h>
#include <iostream>
#include "ShellMITC4.h"

using namespace std;

//#define ACTIVATE_NonlinearShellMITC4_LICENSE(pathToLicense)   NonlinearShellMITC4::path = pathToLicense;

class NonlinearShellMITC4 {
public:

    // constructor
    NonlinearShellMITC4(string path_in, 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);

    // constructor
    NonlinearShellMITC4(string path_in, 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,
                vector<double> &DMatrix_in);


    inline static string path = "";

    // setters & accessors
    //
    size_t get_Rows() { return sizeR_; }
    size_t get_Cols() { return sizeC_; }

    // B-matrix in vector form (in the local csys)
    void get_BMatrix(int igaus_in, int izet_in, double zet_in, vector<double> *BMat_out);

    // local (csys) transposed B-matrix in vector form
    // warning: "get_BMatrix" must be called first to construct the B-Matrix
    // only then the transposed B-Matrix can be called
    void get_BMatrix_Transposed(vector<double> *BMatT) { *BMatT = BMatTransposed_; }

    // determinant of the Jacobian
    //double get_determinant() { return determinant_; }

    // setter for local axes at in-plane GP in case the user decides to load
    // the local axes at the GP (eg for plastic anisotropic csys)
    // the structure for "axesgp_" must be the following:
    // axesgp_in[0:2] = x-local vector
    // axesgp_in[3:5] = y-local vector
    // axesgp_in[6:8] = z-local vector
```

```
    //
    // Note: if local axes are not setted, the API will calculate them
    void set_LocalAxes_All(vector<double> axesgp_in) { axesgp_all_ = axesgp_in; }
    vector<double> get_LocalAxes_All() { return axesgp_all_; }

    // calculates the local axes at each in-plane Gauss point. The local axes at the in-plane GP
    // are located at the mid-surface (zet = 0.0) of the shell so that for non-zero zet coordinate the
    // local axes are the same as at zet = 0.0.
    void Calculate_Local_Axes_All_GP();

    // get the linear Stiffness Matrix
    vector<double> Calculate_StiffnessMITC4();
    vector<double> get_StiffnessMITC4() { return Stiffness_; }

    // get the initial stress (geometric) Stiffness Matrix
    vector<double> get_Stiffness_InitialStress();

    // get the Internal Forces vector
    vector<double> Calculate_Internal_Forces();
    vector<double> get_Internal_Forces() { return Internal_Force_; }

    // calculates the Stiffness and Internal Force vector in one go
    void Calculate_Stiffness_Internal();

    // get the strain tensors
    vector<double> get_StrainsMITC4();
    vector<double> get_StrainsMITC4_Global();

    // rotate axes at the GPs
    void rotate_GP_axes();

    // get the stress tensors
    vector<double> get_StressesMITC4();
    vector<double> get_StressesMITC4_Global();

    //
    void Jacobi(vector<vector<double» &ain, vector<vector<double» &bin,
                vector<double> &eigenvalues_, vector<vector<double» &eigenvectors_);



private:

    //void Permitelemento(string path);

    int igaus_;

    int izet_;

    double zet_;
    double zet_weight_;
    double determinant_; // determinant of the Jacobian

    vector<vector<double» xyz_; // coordinates for the 4-nodes of the shell
    vector<vector<double» vnor_; // normal vector at each node
    vector<double> thick_; // thickness at each node
    vector<vector<double» xlocal_; // x-local nodal csys at each node
    vector<vector<double» ylocal_; // y-local nodal csys at each node
    vector<double> qsiGPcoord_; // qsi coordinate of GP
    vector<double> etaGPcoord_; // eta coordinate of GP
    vector<double> zetGPcoord_; // zet coordinate of GP
    vector<double> qsiGPweigth_; // GP weight along qsi
    vector<double> etaGPweigth_; // GP weight along eta
    vector<double> zetGPweigth_; // GP weight along zet
    vector<vector<double» fformaGP_; // shape functions at each GP
    vector<vector<double» fforma_;
    vector<vector<double» qformaGP_; // qsi shape functions derivatives at each GP
    vector<vector<double» qforma_;
    vector<vector<double» eformaGP_; // eta shape functions derivatives at each GP
    vector<vector<double» eforma_;
    vector<vector<double» zformaGP_; // zet shape functions derivatives at each GP
    vector<double> axesgp_all_; // local axes at all in-plane GPs. The local axes are defined
                                // for in-plane GPs only (zet = 0)
    vector<double> axesgp_; // local axes at the GP
    vector<double> axesgp_all_temp_;

    //
    vector<double> Displacements_;

    //
    vector<double> Strains_;
    vector<double> Strains_Global_;

    //
    // the stresses at the integration points must be given as input during class constructor
    // the stresses are required for the internal force vector and initial stress stiffness matrix calculations
```

```
    // 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_" 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_[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]
    vector<double> stresses_; // stresses at the integration points
    vector<double> stresses_Global; // stresses at the integration point at the Global csys

    //
    // D-Matrix vector with the D-Matrix in vectorial form for all integration points.
    // each integration point has 25 (5x5) positions in the D-Matrix vector
    vector<double> DMatrix_;


    // "BMat_" is in vector form. It is calculated and stored row-wise as follows:
    //      BMat_[0:23]    = B[0,0:24]
    //      BMat_[24:47]   = B[1,0:24]
    //      BMat_[48:71]   = B[2,0:24]
    //      BMat_[72:95]   = B[3,0:24]
    //      BMat_[96:119] = B[4,0:24]
    vector<double> BMat_; // final B-matrix in vector form at the integration point
    vector<double> BMatTransposed_; // transposed B-matrix at the integration point

    //
    size_t sizeR_; // number of rows of the B-matrix
    size_t sizeC_; // number of columns of the B-matrix

    // linear Stiffness Matrix
    vector<double> Stiffness_; // linear Stiffness Matrix
    vector<double> Stiffness_Gauss_;

    // initial stress stiffness matrix
    vector<double> StiffnessInitialStress_; // initial stress stiffness matrix

    // Internal Force Vector
    vector<double> Internal_Force_; // Internal Force Vector
    vector<double> Internal_Force_Gauss_;

    // initialise matrix in vector form
    void initialise_matrix_inVector(size_t lines, size_t columns, vector<double> &mat);

    // cross-product
    vector<double> prodve(vector<double> vec1, vector<double> vec2, int khoice);

    // determinant
    double determinant(vector<vector<double»&jacobian);

    // 3x3 inverse
    void inverse_3x3(vector<vector<double»&racobb, double determ, vector<vector<double»&racobi);

    // norm of a vector
    double norm_vector(vector<double> v_in);

    // initialisation of a matrix
    void initialise_matrix(int lines, int columns, vector<vector<double»& mat);

    // calculates the Jacobian (3x3) matrix at each GP defined by the in-plane GP number "igau"
    // and parametric coordinate along thickness direction "zet"
    vector<vector<double» Jacobian(int igau, double zet);

    // Matrix multiplication
    vector<double> Mat_Mult(size_t sizeR, size_t sizeC, size_t sizeCommon, vector<double> B, vector<double> C);

    vector<double> Mat_Mult_Upper_Triangle(size_t sizeR, size_t sizeC, size_t sizeCommon, vector<double> B, vector<double> C);

    // transpose a matrix in vector form
    vector<double> Mat_Transpose(size_t sizeA_R, size_t sizeA_C, vector<double> A);

    // load shape functions and derivatives
    void Load_Shape_Derivatives(int shift);

    // Linear Stiffness Matrix
    void Stiffness_MITC4();

    // stiffness matrix at a Gauss Point
    vector<double> Stiffness_Gauss_MITC4(vector<double> DMatrix_in);

    // stiffness matrix and internal force vector at a Gauss Point
    void Stiffness_Internal_Gauss_MITC4(vector<double> DMatrix_in, vector<double> Stresses_in);

```

```
    // initial stress stiffness matrix
    void calculate_StiffnessInitialStress();

    // duvw matrix for the initial stress-stiffness matrix
    vector<vector<double» duvw_Gauss_point(vector<vector<double» racobi);

    // Internal Force vector
    void calculate_Internal_Forces();

    //
    void StrainsMITC4();
    void Strains_Transform();

    //
    vector<vector<double» rnstrain(vector<vector<double» fk);

    void StressesMITC4();
    void Stresses_Transform();

};


```