KomplekseSolver

Generated by Doxygen 1.8.7

Tue Jun 10 2014 13:05:30

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Element Class Reference

Element (p. 3) contains element data and relevant functions.

`#include <Element.h>`

Public Member Functions

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW **Element** ()

  < *Neccessary for std::vector of objects with Eigen members.*
- double **xOfZeta** (double zeta1, double zeta2)
- double **yOfZeta** (double zeta1, double zeta2)
- void **calculateArea** ()
- void **calculateAreaCentre** ()
- double **getSecondAreaMomentX** (double)
- double **getSecondAreaMomentY** (double)
- double **getSecondAreaMomentProduct** (double, double)
- MatrixXd **getShapefunction** (double zeta1, double zeta2, double zeta3)
- MatrixXd **getShapefunction** (double x, double y)
- MatrixXd **getNMatrix** (double zeta1, double zeta2, double zeta3)

  Returns a matrix of matrices with shape functions.
- MatrixXd **getNMatrix** (double x, double y)
- void **calculateConstants** ()

  Calculates the constant element matrices.
- std::vector< int > **getIEG** ()
- void **calculateStiffnessMatrix** ()
- void **calculateElementLoadTorsion** ()

  Calculates the initial load vector for torsion.
- void **calculateElementLoadShearX** ()

  Calculates the initial load vector for shear along X.
- void **calculateElementLoadShearY** ()

  Calculates the initial load vector for shear along Y.
- void **setElementValue** (MatrixXd &system, MatrixXd &local)
- void **computeTau** ()

Public Attributes

- int **i**

  Node *(*p. *12) 1 in the element.*
- int **j**

  Node *(*p. *12) 2 in the element.*
- int **k**

  Node *(*p. *12) 3 in the element.*
- double **area**

  Elements total area.
- double **material**

  Young's modulus for the element.
- double **shearModule**
- double **Iex**

  Elements second area moment x.
- double **Iey**

  Elements second area moment y.
- double **Iexy**

  Elements second area moment product.
- double **xl1**
- double **xl2**
- double **xl3**

  Distance in x direction from node to element area center.
- double **yl1**
- double **yl2**
- double **yl3**

  Distance in y direction from node to element area center.
- double **ecx**
- double **ecy**

  Elements area centre.
- double **dx**
- double **dy**

  Distance between mesh- and node area centre.
- double **x1**
- double **x2**
- double **x3**

  Elements nodepositions in x-directions.
- double **y1**
- double **y2**
- double **y3**

  Elements nodepositions in x-directions.
- double **xp1**
- double **xp2**
- double **xp3**

  Nodal x-coordinates in principal axes.
- double **yp1**
- double **yp2**
- double **yp3**

  Nodal y-coordinates in principal axes.
- MatrixXd **B**

  2X3 Matrix containing differentials of shapefunction with respect to x and y respectively
- MatrixXd **G**

Matrix with shearmodule used to determine initial element loads in tortional analysis.

- MatrixXd **XoverY**

  6x1 matrix with x1, x2, x3, y1, y2 and y3 of the element.

- MatrixXd **xCoordinates**

  1x3 matrix with x-coordinates.

- MatrixXd **yCoordinates**

  1X3 matrix with y-coordinates.

- MatrixXd **elementStiffness**

  Matrix used to temporarily hold element stiffnes coefficients.

- MatrixXd **elementLoad**

  Matrix used to temporarily hold element load coefficients.

- MatrixXd **elementDisplacement**

  Matrix used to temporarily hold element displacement values.

- MatrixXd **initialStrain**

  Matrix used to temporarily hold element initial strain coefficients.

- std::vector< MatrixXd > **tau**

  Vector temporarily storing shear stresses during calculation.

2.1.1 Detailed Description

Element (p. 3) contains element data and relevant functions.

2.1.2 Member Function Documentation

2.1.2.1 void Element::calculateArea ( )

Calculates the area of the element.
$A_e = \frac{1}{2} \cdot |J|$

2.1.2.2 void Element::calculateAreaCentre ( )

Calculates the area centre of the element.
$ec_x = \frac{x1+x2+x3}{3}$
$ec_y = \frac{y1+y2+y3}{3}$

2.1.2.3 void Element::computeTau ( )

Calculates shear stresses at nodes using initialStrain and elementsDisplacement
$\tau = Bv + \varepsilon_0$

2.1.2.4 std::vector< int > Element::getIEG ( )

Creates a vector with node numbers that makes up the element.

Returns

Vector<Int> containing element nodes.

2.1.2.5 double Element::getSecondAreaMomentProduct (double *Ax*, double *Ay*)

Calculating the product of moment of inertia

$dx_e = Ac_x - Ac_{ex}$
$dy_e = Ac_y - Ac_{ey}$
$I_{xy} = \sum (I_{xy_e} + dx_e * dy_e * A_e)$

2.1.2.6 double Element::getSecondAreaMomentX (double *Ay*)

Calculates the second moment of area for the element.

Parameters

Ay	Area centre of the mesh

$dy_e = A_y - Ac_{ey}$
$I_x = \sum (I_{x_e} + dy_e^2 * A_e)$

2.1.2.7 double Element::getSecondAreaMomentY (double *Ax*)

Calculates the second moment of area for the element.

Parameters

Ax	Area centre of the mesh

$dx_e = A_x - Ac_{ex}$
$I_y = \sum (I_{y_e} + dx_e^2 * A_e)$

2.1.2.8 MatrixXd Element::getShapefunction (double *zeta1*, double *zeta2*, double *zeta3*)

Calculates the Matrix containing shape fucntions

Returns

MatrixXd(1,3) with shape functions

2.1.2.9 MatrixXd Element::getShapefunction (double *x*, double *y*)

Returns the shapefunction as a function of x and y. $\zeta_1 = y_{23}x + x_{32}y + (x_2y_3 - x_3y_2)$
$\zeta_1 = y_{31}x + x_{13}y + (x_3y_1 - x_1y_3)$
$\zeta_1 = y_{12}x + x_{21}y + (x_1y_2 - x_2y_1)$

Returns

MatrixXd(1, 3) with shape function.

2.1.2.10 void Element::setElementValue (MatrixXd & *system*, MatrixXd & *local*)

Uses **getIEG()** (p. 5) to assign system values to corresponding local parameters.

2.1.2.11 double Element::xOfZeta (double *zeta1*, double *zeta2*)

Returns the x-coordinate of a point in the element with respect to its area center as a function of area coordinates.
$x = (x_{l1} - x_{l3})\zeta_1 + (x_{l2} - x_{l3})\zeta_2 + x_{l3}$

2.1.2.12 double Element::yOfZeta (double *zeta1*, double *zeta2*)

Returns the x- and y-coordinates of a point in the element with respect to its area center as a function of area coordinates.
$y = (y_{l1} - y_{l3})\zeta_1 + (y_{l2} - y_{l3})\zeta_2 + y_{l3}$

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Element.h
- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Element.cpp

2.2 Mesh Class Reference

Mesh (p. 7) contains mesh data and has function to calculate and retrieve mesh properties.

`#include <Mesh.h>`

Public Member Functions

- **Mesh** (std::vector< **Element** >, int, std::vector< **Node** >, int, double)
- void **solveBendingMoments** ()
- void **solveTorsion** ()

  Calculates the shear stress distribution under torsional load.
- void **solveShearX** ()

  Calculates the shear stress distribution under shear force along x-axis.
- void **solveShearY** ()

  Calculates the shear stress distribution under shear force along y-axis.
- void **setEModule** (double)
- void **calculateMeshProperties** ()
- void **transformCoordinates** ()
- void **calculateSecondAreaMoments** ()
- void **calculateAnglePrincipalAxes** ()
- double **getAnglePrincipalAxes** ()
- void **calculateLineCenterNodes** ()
- void **solveEquations** (SparseMatrix< double > &A, MatrixXd &b, MatrixXd &C)
- void **constrainNodesNearCentre** (int)
- void **constrainNode** (SparseMatrix< double > &K, MatrixXd &R, int nNumber)
- **Node getNodeAt** (int n)
- **Element getElementAt** (int e)

Public Attributes

- EIGEN_MAKE_ALIGNED_OPERATOR_NEW int **nElem**

  < Neccessary for std::vector of objects with Eigen members.
- int **nNode**

  Number of nodes.
- double **Ax**

  Area centre for mesh in X-direction.
- double **Ay**

  Area centre for mesh in Y-direction.
- double **totalArea**

Total area of the mesh.

- double **areaStiffness**

  Sum of all Ai ∗ Ei.

- double **Ix**

  Second area moment for mesh in X-direction.

- double **Iy**

  Second area moment for mesh in Y-direction.

- double **Ixy**

  Second area moment for mesh in XY-direction.

- double **Imax**

  Maximum second area moment for mesh along principal axis.

- double **anglePrincipalAxes**

  Stores the angle of the principal axes in degrees.

- double **anglePrincipalAxesRad**

  Stores the angle of the principal axes in radians.

- MatrixXd **sigmaAxial**

  Stores nodal stress from axial force.

- MatrixXd **sigmaOfX**

  Stores all nodal stresses from moment about x-axis.

- MatrixXd **sigmaOfY**

  Stores all nodal stresses from moment about y-axis.

- MatrixXd **tauXZtorsion**

  Stores nodal τ_{xz} from torsional load.

- MatrixXd **tauYZtorsion**

  Stores nodal τ_{yz} from torsional load.

- MatrixXd **tauRtorsion**

  Stores nodal τ_R from torsional load.

- MatrixXd **tauXZshearX**

  Stores nodal τ_{xz} from shear load in x-direction.

- MatrixXd **tauYZshearX**

  Stores nodal τ_{yz} from shear load in x-direction.

- MatrixXd **tauRshearX**

  Stores nodal τ_R from shear load in x-direction.

- MatrixXd **tauXZshearY**

  Stores nodal τ_{xz} from shear load in y-direction.

- MatrixXd **tauYZshearY**

  Stores nodal τ_{yz} from shear load in y-direction.

- MatrixXd **tauRshearY**

  Stores nodal τ_R from shear load in y-direction.

- MatrixXd **printTemp1**

  Used in testing to store desired value for printing.

- MatrixXd **printTemp2**

  Used in testing to store desired value for printing.

2.2.1 Detailed Description

Mesh (p. 7) contains mesh data and has function to calculate and retrieve mesh properties.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Mesh::Mesh (std::vector< Element > *elements,* int *nElem,* std::vector< Node > *nodes,* int *nNode,* double *material*)

Constructor computes the area, areacentre and the stiffness matrix

Parameters

elements	List of all elements
nodes	List of all nodes

2.2.3 Member Function Documentation

2.2.3.1 void Mesh::calculateAnglePrincipalAxes ( )

Calculates the angle of one of the inertial axis.
$t = \frac{-2I_{xy}}{I_x - I_y}$
$\alpha = \frac{1}{2}\arctan(t)\frac{180}{\pi}$

Returns

Angle of one inertial axis.

2.2.3.2 void Mesh::calculateMeshProperties ( )

Function run by constructor. It runs through all elements calculating element areas and element centres for each element.
It also calculates total area of mesh, and area centre of the mesh using
$A_x = \sum(\frac{dx_e * a_e}{A})$ and $A_y = \sum(\frac{dy_e * a_e}{A})$

2.2.3.3 void Mesh::calculateSecondAreaMoments ( )

Loops through all elements calculating the distance between mesh- and element area centres, and calculates the second moment of inertia for the mesh's x-, y- and xy-axis.

2.2.3.4 void Mesh::setEModule (double *mat*)

Sets an uniform E-module for the material over the whole mesh.

2.2.3.5 void Mesh::solveBendingMoments ( )

Calculates the resulting stresses from unit bending moments
$\sigma = dy/I_x$ $\sigma = dx/I_y$

2.2.3.6 void Mesh::solveEquations (SparseMatrix< double > & *A*, MatrixXd & *b*, MatrixXd & *C*)

Solves the eq
$Kr = R$,
where K is the stiffness matrix, and r and R are vectors with displacements and loads in each node.

2.2.3.7 void Mesh::transformCoordinates ( )

Transforms coordinates for computation purposes.
$x' = x - A_x cos(\alpha) + y - A_y sin(\alpha)$
$x' = y - A_y cos(\alpha) - x - A_x sin(\alpha)$

2.2.4 Member Data Documentation

2.2.4.1 EIGEN_MAKE_ALIGNED_OPERATOR_NEW int Mesh::nElem

< Neccessary for std::vector of objects with Eigen members.

Number of elements

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Mesh.h
- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Mesh.cpp

2.3 MSHParser Class Reference

MSHParser (p. 11) interpret meshes structured in the ".msh" format.

`#include <MSHParser.h>`

Public Member Functions

- **MSHParser** (std::string)
- std::vector< **Element** > **getElements** ()
- std::vector< **Node** > **getNodes** ()
- int **getNumberOfNodes** ()
- int **getNumberOfElements** ()
- **Element getElementAt** (int)
- **Node getNodeAt** (int)

2.3.1 Detailed Description

MSHParser (p. 11) interpret meshes structured in the ".msh" format.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 MSHParser::MSHParser (std::string *fileName*)

Parameters

String	Filename of the file to parse. Excluding the file type (ending)

2.3.3 Member Function Documentation

2.3.3.1 Element MSHParser::getElementAt (int *e*)

Returns

Element (p. 3) at the given position

2.3.3.2 std::vector< Element > MSHParser::getElements ( )

Returns

vector<Element> containing all valid elements parsed from file

2.3.3.3 Node MSHParser::getNodeAt (int *n*)

Returns

Node (p. 12) at the given position

2.3.3.4 std::vector< Node > MSHParser::getNodes ( )

Returns

vector<Node> containing all nodes parsed from file

2.3.3.5 int MSHParser::getNumberOfElements ( )

Returns

number of elements

2.3.3.6 int MSHParser::getNumberOfNodes ( )

Returns

number of nodes

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/MSHParser.h
- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/MSHParser.cpp

2.4 Node Struct Reference

Struct that holds the x and y coordinate for each node.

`#include <Node.h>`

Public Attributes

- std::vector< int > **depElem**

  std::vector containing a list of connected elements.
- std::vector< int > **depIEG**

  std::vector containing internal numbering of node in connected elements.
- double **x**
- double **y**
- double **xp**

- double **yp**
- double **sigmaAxial**

  Stress due to axial force.
- double **NofX**
- double **NofY**

  Normal stresses due to Mx and My.
- MatrixXd **tau**

  2x1 matrix containing tauXZ and tauYZ.

2.4.1 Detailed Description

Struct that holds the x and y coordinate for each node.

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Node.h

2.5 ResultWriter Class Reference

Opens and writes results to file.

`#include <ResultWriter.h>`

Public Member Functions

- **ResultWriter** (std::string, **Mesh** mesh)
- bool **open** ()
- void **save** ()
- void **writeNormalStress** ()
- void **writeMomentStress** (std::vector< **Node** >, int)
- void **writeInertialAxes** ()
- void **writeMeshAreaProperties** ()
- void **writeScalarData** (std::string, std::vector< double >)
- void **writeSingleValue** (std::string, double)
- void **writeBoundaryNodes** (int, std::vector< int >)
- void **writeVtkHeader** ()
- void **writeMeshTopology** (**Mesh** &mesh, MatrixXd data)
- void **writeMeshTopology** (**Mesh** &mesh)
- void **writeMeshTopology** (int nNode, std::vector< **Node** >, int nElem, std::vector< **Element** >)
- void **writePointScalarData** (std::string name, MatrixXd data)
- void **writePointVectorData** (std::string name, MatrixXd data)

2.5.1 Detailed Description

Opens and writes results to file.

Resultwriter is responsible for opening and write data to file. It also contains definitions and the structure of results. Most of the calculations is done in **Mesh** (p. 7) and **Solver** (p. 15), and only calculations regarding mapping a force to a specific element or node performed in this class.

2.5.2 Member Function Documentation

2.5.2.1 void ResultWriter::writeInertialAxes ( )

Writes the angle of one of the inertial axes to file.

Result tag "#InertialAxes"

2.5.2.2 void ResultWriter::writeMeshAreaProperties ( )

Writes the mesh properties to result file.

Result tag "#MeshProperties"

Properties written:

- Area centre,
- Total area,
- Shear centre,
- Moment of inertia-X,
- Moment of inertia-Y,
- Moment of inertia product

2.5.2.3 void ResultWriter::writeMomentStress (std::vector< Node > *nodes,* int *nLength*)

Calculate stresses caused by an unit moment about the inertal axis

Result tag "#Sigma due to M_x"
Result tag "#Sigma due to M_y"

It first creates a unit vector along the inertial axis. Using vector algebra to calculate the distance from the inertial axis to the each node.
$y = ||(a \vec{-} p) - ((a \vec{-} p) \cdot \vec{n})\vec{n}||$

Force in a node is given by
$\sigma = \frac{M}{I} \cdot y$,
where y is the distance calculated above.
The result written by this method is based on a unit moment(M = 1).

Parameters

Vector<Node>	containing stresses in x,y direction for all nodes
Integer	Number of nodes in vector
Mesh *(p. 7)*	reference to mesh class containing the data

2.5.2.4 void ResultWriter::writeNormalStress ( )

Writes a result containing forces in each nodes given an uniform normal stress over the mesh.

Result tag "#Sigma due to N"

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/ResultWriter.h
- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/ResultWriter.cpp

2.6 Solver Class Reference

Contains the main function and initializes **MSHParser** (p. 11), **Mesh** (p. 7), and finally **ResultWriter** (p. 13).

`#include <Solver.h>`

Public Member Functions

- **Solver** (std::string, double material)

2.6.1 Detailed Description

Contains the main function and initializes **MSHParser** (p. 11), **Mesh** (p. 7), and finally **ResultWriter** (p. 13).

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

- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Solver.h
- D:/Dokumenter/Fordypningsprosjekt/Kode/Prosjektoppgave Komplekse tverrsntt/KomplekseTverrsnitt/↩ Solver/Solver.cpp