



An Object Oriented Finite Element Library

Reference Guide

Release 3.0.0

Rachid Touzani
Laboratoire de Mathématiques
Université Blaise Pascal (Clermont–Ferrand)
63177 Aubière, France
e-mail : Rachid.Touzani@univ-bpclermont.fr

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Conservation Law Equations	15
Electromagnetics	16
Finite Element Mesh	17
Fluid Dynamics	30
General Purpose Equations	31
Global Variables	35
Heat Transfer	38
Input/Output	39
Interface Problems	40
Laplace equation	41
OFELI	42
Physical properties of media	43
Shape Function	44
Solid Mechanics	45
Solver	46
Utilities	63
Vector and Matrix	83

Chapter 2

Namespace Index

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AbsEqua< complex_t >	111
Equation< complex_t, NEN_, NEE_, NSN_, NSE_ >	466
Equa.Electromagnetics< complex_t, 3, 3, 2, 2 >	398
EC2D1T3	298
HelmholtzBT3	501
AbsEqua< double >	111
Equation< double, NEN_, NEE_, NSN_, NSE_ >	466
Equa.Fluid< double, 3, 6, 2, 4 >	410
TINS2DT3B	812
Equa.Solid< double, 4, 12, 3, 9 >	436
Elas3DT4	373
Equa.Solid< double, 8, 24, 4, 12 >	436
Elas3DH8	358
AbsEqua< real_t >	111
Equation< real_t, NEN_, NEE_, NSN_, NSE_ >	466
Equa.Electromagnetics< real_t, 3, 6, 2, 4 >	398
Equa.Fluid< real_t, 4, 8, 2, 4 >	410
NSP2DQ41	712
Equa.Laplace< real_t, 2, 2, 1, 1 >	423
Laplace1DL2	548
Equa.Laplace< real_t, 3, 3, 1, 1 >	423
Laplace1DL3	564
Equa.Laplace< real_t, 3, 3, 2, 2 >	423
Laplace2DFVT	576
Laplace2DMHRT0	590
Laplace2DT3	605
Equa.Solid< real_t, 2, 12, 1, 1 >	436
Beam3DL2	129
Equa.Solid< real_t, 2, 4, 1, 2 >	436
Bar2DL2	114
Equa.Solid< real_t, 3, 6, 2, 4 >	436
Elas2DT3	338
Equa.Solid< real_t, 4, 8, 2, 4 >	436

Elas2DQ4	320
Equa_Therm< real_t, 2, 2, 1, 1 >	452
DC1DL2	168
Equa_Therm< real_t, 3, 3, 2, 2 >	452
DC2DT3	186
DC3DAT3	226
Equa_Therm< real_t, 4, 4, 3, 3 >	452
DC3DT4	245
Equa_Therm< real_t, 6, 6, 3, 3 >	452
DC2DT6	208
AbsEqua< T_ >	111
Equation< T_, NEN_, NEE_, NSN_, NSE_ >	466
Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ >	398
Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ >	410
Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ >	423
Equa_Solid< T_, NEN_, NEE_, NSN_, NSE_ >	436
Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ >	452
BiotSavart	147
Domain	281
Edge	311
EdgeList	314
Eigen	315
Element	388
ElementList	395
Estimator	479
FastMarching2D	480
FEShape	481
Hexa8	515
Line2	638
Line2H	641
Line3	643
Penta6	727
Quad4	744
Tetra4	809
triangle	834
Triang3	826
Triang6S	828
Figure	483
Brick	162
Circle	165
Ellipse	396
Polygon	736
Rectangle	747
Sphere	786
Triangle	??
FMM2D	485
FMM3D	487
FMMSolver	489
Funct	490
Gauss	492
Grid	492
IOField	536

IPF	539
Iter< T_ >	547
LocalMatrix< T_, NR_, NC_ >	644
LocalVect< T_, N_ >	651
Material	655
Matrix< T_ >	658
BMatrix< T_ >	151
DMatrix< T_ >	265
DSMatrix< T_ >	285
SkMatrix< T_ >	757
SkSMatrix< T_ >	771
SpMatrix< T_ >	789
TrMatrix< T_ >	837
Mesh	668
MeshAdapt	688
Muscl	692
Muscl1D	696
ICPG1D	517
LCL1D	620
Muscl2DT	699
ICPG2DT	524
LCL2DT	625
Muscl3DT	703
ICPG3DT	531
LCL3DT	632
Node	707
NodeList	711
Partition	726
PhaseChange	729
Point< T_ >	730
Point2D< T_ >	733
Prec< T_ >	739
Precond< T_ >	742
Prescription	742
Reconstruction	746
Side	749
SideList	756
SteklovPoincare2DBE	805
Tabulation	807
Timer	811
UserData< T_ >	849
Vect< T_ >	850
Point< real_t >	730
Vect< double >	850

Chapter 4

Class Index

4.1 Class List

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

AbsEqua< T_ >	Mother abstract class to describe equation	111
Bar2DL2	To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node	114
Beam3DL2	To build element equations for 3-D beam equations using 2-node lines	129
BiotSavart	Class to compute the magnetic induction from the current density using the Biot--Savart formula	147
BMatrix< T_ >	To handle band matrices	151
Brick	To store and treat a brick (parallelepiped) figure	162
Circle	To store and treat a circular figure	165
DC1DL2	Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements	168
DC2DT3	Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles	186
DC2DT6	Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles	208
DC3DAT3	Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles	226
DC3DT4	Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra	245
DMatrix< T_ >	To handle dense matrices	265
Domain	To store and treat finite element geometric information	281

DSMatrix< T_ >	To handle symmetric dense matrices	285
EC2D1T3	Eddy current problems in 2-D domains using solenoidal approximation	298
Edge	To describe an edge	311
EdgeList	Class to construct a list of edges having some common properties	314
Eigen	Class to find the smallest eigenvalues and corresponding eigenvectors in a generalized eigenproblem using the Bathe subspace iteration method	315
Elas2DQ4	To build element equations for 2-D linearized elasticity using 4-node quadrilaterals	320
Elas2DT3	To build element equations for 2-D linearized elasticity using 3-node triangles	338
Elas3DH8	To build element equations for 3-D linearized elasticity using 8-node hexahedra	358
Elas3DT4	To build element equations for 3-D linearized elasticity using 4-node tetrahedra	373
Element	To store and treat finite element geometric information	388
ElementList	Class to construct a list of elements having some common properties	395
Ellipse	To store and treat an ellipsoidal figure	396
Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for Electromagnetics Equation classes	398
Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for Fluid Dynamics Equation classes	410
Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for classes about the Laplace equation	423
Equa_Solid< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for Solid Mechanics Finite Element classes	436
Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for Heat transfer Finite Element classes	452
Equation< T_, NEN_, NEE_, NSN_, NSE_ >	Abstract class for all equation classes	466
Estimator	To calculate an a posteriori estimator of the solution	479
FastMarching2D	To run a Fast Marching Method on 2-D structured uniform grids	480
FEShape	Parent class from which inherit all finite element shape classes	481
Figure	To store and treat a figure (or shape) information	483
FMM2D	Class for the fast marching 2-D algorithm	485
FMM3D	Class for the 3-D fast marching algorithm	487
FMMSolver	The Fast Marching Method solver	489
Funct	A simple class to parse real valued functions	490

Gauss	Calculate data for Gauss integration	492
Grid	To manipulate structured grids	492
HelmholtzBT3	Builds finite element arrays for Helmholtz equations in a bounded media using 3-- Node triangles	501
Hexa8	Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation	515
ICPG1D	Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D	517
ICPG2DT	Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D	524
ICPG3DT	Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D	531
IOField	Enables working with files in the XML Format	536
IPF	To read project parameters from a file in IPF format	539
Iter< T_ >	Class to drive an iterative process	547
Laplace1DL2	To build element equation for a 1-D elliptic equation using the 2-Node line element (P_1)	548
Laplace1DL3	To build element equation for the 1-D elliptic equation using the 3-Node line (P_2)	564
Laplace2DFVT	To build and solve the Laplace equation using a standard Finite Volume method	576
Laplace2DMHRT0	To build element equation for the 2-D elliptic equation using the Mixed Hybrid finite element at lowest degree (Raviart-Thomas RT_0)	590
Laplace2DT3	To build element equation for the Laplace equation using the 2-D triangle element (P_1)	605
LCL1D	Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme	620
LCL2DT	Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles	625
LCL3DT	Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra	632
Line2	To describe a 2-Node planar line finite element	638
Line2H	To describe a 2-Node Hermite planar line finite element	641
Line3	To describe a 3-Node quadratic planar line finite element	643

LocalMatrix< T_, NR_, NC_ >	Handles small size matrices like element matrices, with a priori known size	644
LocalVect< T_, N_ >	Handles small size vectors like element vectors	651
Material	To treat material data. This class enables reading material data in material data files. It also returns these informations by means of its members	655
Matrix< T_ >	Virtual class to handle matrices for all storage formats	658
Mesh	To store and manipulate finite element meshes	668
MeshAdapt	To adapt mesh in function of given solution	688
Muscl	Parent class for hyperbolic solvers with Muscl scheme	692
Muscl1D	Class for 1-D hyperbolic solvers with Muscl scheme	696
Muscl2DT	Class for 2-D hyperbolic solvers with Muscl scheme	699
Muscl3DT	Class for 3-D hyperbolic solvers with Muscl scheme using tetrahedra	703
Node	To describe a node	707
NodeList	Class to construct a list of nodes having some common properties	711
NSP2DQ41	Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using Q_1/P_0 element and a penalty formulation for the incompressibility condition	712
Partition	To partition a finite element mesh into balanced submeshes	726
Penta6	Defines a 6-node pentahedral finite element using P_1 interpolation in local coordinates $(s.x, s.y)$ and Q_1 isoparametric interpolation in local coordinates $(s.x, s.z)$ and $(s.y, s.z)$	727
PhaseChange	This class enables defining phase change laws for a given material	729
Point< T_ >	Defines a point with arbitrary type coordinates	730
Point2D< T_ >	Defines a 2-D point with arbitrary type coordinates	733
Polygon	To store and treat a polygonal figure	736
Prec< T_ >	To set a preconditioner	739
Precond< T_ >	An abstract class from which derive all preconditioning classes	742
Prescription	To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable	742
Quad4	Defines a 4-node quadrilateral finite element using Q_1 isoparametric interpolation	744

Reconstruction	To perform various reconstruction operations	746
Rectangle	To store and treat a rectangular figure	747
Side	To store and treat finite element sides (edges in 2-D or faces in 3-D)	749
SideList	Class to construct a list of sides having some common properties	756
SkMatrix< T_ >	To handle square matrices in skyline storage format	757
SkSMMatrix< T_ >	To handle symmetric matrices in skyline storage format	771
Sphere	To store and treat a sphere	786
SpMatrix< T_ >	To handle matrices in sparse storage format	789
SteklovPoincare2DBE	Solver of the Steklov Poincare problem in 2-D geometries using piecewise constant boundary elements	805
Tabulation	To read and manipulate tabulated functions	807
Tetra4	Defines a three-dimensional 4-node tetrahedral finite element using P_1 interpolation	809
Timer	To handle elapsed time counting	811
TINS2DT3B	Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles	812
Triang3	Defines a 3-Node (P_1) triangle	826
Triang6S	Defines a 6-Node straight triangular finite element using P_2 interpolation	828
Triangle	To store and treat a triangle	??
triangle	Defines a triangle. The reference element is the rectangle triangle with two unit edges	834
TrMatrix< T_ >	To handle tridiagonal matrices	837
UserData< T_ >	Abstract class to define by user various problem data	849
Vect< T_ >	To handle general purpose vectors	850

Chapter 5

Module Documentation

5.1 Conservation Law Equations

For solvers of conservation law equations.

Classes

- class [ICPG1D](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.
- class [ICPG2DT](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.
- class [ICPG3DT](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D.
- class [LCL1D](#)
Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.
- class [LCL2DT](#)
Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.
- class [LCL3DT](#)
Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.
- class [Muscl](#)
Parent class for hyperbolic solvers with Muscl scheme.
- class [Muscl1D](#)
Class for 1-D hyperbolic solvers with [Muscl](#) scheme.
- class [Muscl2DT](#)
Class for 2-D hyperbolic solvers with [Muscl](#) scheme.
- class [Muscl3DT](#)
Class for 3-D hyperbolic solvers with [Muscl](#) scheme using tetrahedra.

5.1.1 Detailed Description

For solvers of conservation law equations.

5.2 Electromagnetics

Gathers electromagnetic equation classes.

Classes

- class [BiotSavart](#)
Class to compute the magnetic induction from the current density using the Biot-Savart formula.
- class [EC2D1T3](#)
Eddy current problems in 2-D domains using solenoidal approximation.
- class [Equa_Electromagnetics](#)< [T_](#), [NEN_](#), [NEE_](#), [NSN_](#), [NSE_](#) >
Abstract class for Electromagnetics [Equation](#) classes.
- class [HelmholtzBT3](#)
Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.

5.2.1 Detailed Description

Gathers electromagnetic equation classes.

5.3 Finite Element Mesh

Gathers mesh related classes.

Classes

- class [Domain](#)
To store and treat finite element geometric information.
- class [Edge](#)
To describe an edge.
- class [Element](#)
To store and treat finite element geometric information.
- class [Figure](#)
To store and treat a figure (or shape) information.
- class [Rectangle](#)
To store and treat a rectangular figure.
- class [Brick](#)
To store and treat a brick (parallelepiped) figure.
- class [Circle](#)
To store and treat a circular figure.
- class [Sphere](#)
To store and treat a sphere.
- class [Ellipse](#)
To store and treat an ellipsoidal figure.
- class [Triangle](#)
To store and treat a triangle.
- class [Polygon](#)
To store and treat a polygonal figure.
- class [Grid](#)
To manipulate structured grids.
- class [Mesh](#)
To store and manipulate finite element meshes.
- class [MeshAdapt](#)
To adapt mesh in function of given solution.
- class [NodeList](#)
Class to construct a list of nodes having some common properties.
- class [ElementList](#)
Class to construct a list of elements having some common properties.
- class [SideList](#)
Class to construct a list of sides having some common properties.
- class [EdgeList](#)
Class to construct a list of edges having some common properties.
- class [Node](#)
To describe a node.
- class [Partition](#)
To partition a finite element mesh into balanced submeshes.
- class [Side](#)
To store and treat finite element sides (edges in 2-D or faces in 3-D)

Macros

- `#define GRAPH_MEMORY 1000000`
Memory necessary to store matrix graph.
- `#define MAX_NB_SUBMESHES 500`
Maximum number of mesh partitions.
- `#define MAX_NB_ELEMENTS 10000`
Maximal Number of elements.
- `#define MAX_NB_NODES 10000`
Maximal number of nodes.
- `#define MAX_NB_SIDES 30000`
Maximal number of sides in.
- `#define MAX_NB_EDGES 30000`
Maximal Number of edges.
- `#define MAX_NBDOF_NODE 6`
Maximum number of DOF supported by each node.
- `#define MAX_NBDOF_SIDE 6`
Maximum number of DOF supported by each side.
- `#define MAX_NBDOF_EDGE 2`
Maximum number of DOF supported by each edge.
- `#define MAX_NB_ELEMENT_NODES 20`
Maximum number of nodes by element.
- `#define MAX_NB_ELEMENT_EDGES 10`
Maximum number of edges by element.
- `#define MAX_NB_SIDE_NODES 9`
Maximum number of nodes by side.
- `#define MAX_NB_ELEMENT_SIDES 8`
Maximum number of sides by element.
- `#define MAX_NB_ELEMENT_DOF 27`
Maximum number of dof by element.
- `#define MAX_NB_SIDE_DOF 4`
Maximum number of dof by side.
- `#define MAX_NB_INT_PTS 20`
Maximum number of integration points in element.
- `#define MAX_NB_VERTICES 2000`
Maximum number of vertices.
- `#define MAX_NB_HOLES 10`
Maximum number of holes.
- `#define MAX_NB_LINES 200`
Maximum number of lines.
- `#define MAX_NB_LINE_NODES 1000`
Maximum number of nodes on each line.
- `#define MAX_NB_CONT_LINES 50`
Maximum number of lines on each contour.
- `#define MAX_NB_SUB_DOMAINS 5`
Maximum number of subdomains.
- `#define MAX_NB_MATERIALS 10`

Maximum number of materials.

- `#define TheNode (*theNode)`
- `#define TheElement (*theElement)`
- `#define TheSide (*theSide)`
- `#define TheEdge (*theEdge)`
- `#define MeshElements(mesh) for ((mesh).topElement(); (theElement=(mesh).getElement());)`
- `#define MeshActiveElements(mesh) for ((mesh).topElement(); (theElement=(mesh).getActiveElement());)`
- `#define MeshNodeLoop(mesh, node) for ((mesh).topNode(); ((node)=(mesh).getNode());)`
- `#define MeshNodes(mesh) for ((mesh).topNode(); (theNode=(mesh).getNode());)`
- `#define MeshBoundaryNodes(mesh) for ((mesh).topBoundaryNode(); (theNode=(mesh).getBoundaryNode());)`
- `#define MeshSides(mesh) for ((mesh).topSide(); (theSide=(mesh).getSide());)`
- `#define MeshSideSet(sl) for ((sl).top(); (theSide=(sl).get());)`
- `#define MeshBoundarySides(mesh) for ((mesh).topBoundarySide(); (theSide=(mesh).getBoundarySide());)`
- `#define MeshEdges(mesh) for ((mesh).topEdge(); (theEdge=(mesh).getEdge());)`
- `#define theNodeLabel theNode->n()`
- `#define theSideLabel theSide->n()`

A macro that returns side label in a loop using macro MeshSides

- `#define theSideNodeLabel(i) theSide->getNodeLabel(i)`

A macro that returns label of i-th node of side using macro MeshSides

- `#define theElementLabel theElement->n()`

A macro that returns element label in a loop using macro MeshElements

- `#define theElementNodeLabel(i) theElement->getNodeLabel(i)`

A macro that returns label of i-th node of element using macro MeshElements

Functions

- `ostream & operator<< (ostream &s, const Edge &ed)`

Output edge data.

- `ostream & operator<< (ostream &s, const Element &el)`

Output element data.

- `Figure operator&& (const Figure &f1, const Figure &f2)`

Function to define a Figure instance as the intersection of two Figure instances.

- `Figure operator- (const Figure &f1, const Figure &f2)`

Function to define a Figure instance as the set subtraction of two Figure instances.

- `ostream & operator<< (ostream &s, const Material &m)`

Output material data.

- `ostream & operator<< (ostream &s, const Mesh &ms)`

Output mesh data.

- `ostream & operator<< (ostream &s, const MeshAdapt &a)`

Output MeshAdapt class data.

- `ostream & operator<< (ostream &s, const NodeList &nl)`

Output NodeList instance.

- `ostream & operator<< (ostream &s, const ElementList &el)`

Output ElementList instance.

- `ostream & operator<< (ostream &s, const SideList &sl)`

Output *SideList* instance.

- ostream & operator<< (ostream &s, const EdgeList &el)

Output *EdgeList* instance.

- void DeformMesh (Mesh &mesh, const Vect< real_t > &u, real_t a=1)

Calculate deformed mesh using a displacement field.

- void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real_t > &u1, Vect< real_t > &u2, size_t nx, size_t ny=0, size_t nz=0, size_t dof=1)

Function to redefine a vector defined on a mesh to a new mesh.

- void MeshToMesh (Mesh &m1, Mesh &m2, const Vect< real_t > &u1, Vect< real_t > &u2, const Point< real_t > &xmin, const Point< real_t > &xmax, size_t nx, size_t ny, size_t nz, size_t dof=1)

Function to redefine a vector defined on a mesh to a new mesh.

- real_t getMaxSize (const Mesh &m)

Return maximal size of element edges for given mesh.

- real_t getMinSize (const Mesh &m)

Return minimal size of element edges for given mesh.

- real_t getMinElementMeasure (const Mesh &m)

Return minimal measure (length, area or volume) of elements of given mesh.

- real_t getMinSideMeasure (const Mesh &m)

Return minimal measure (length or area) of sides of given mesh.

- real_t getMaxSideMeasure (const Mesh &m)

Return maximal measure (length or area) of sides of given mesh.

- real_t getMeanElementMeasure (const Mesh &m)

Return average measure (length, area or volume) of elements of given mesh.

- real_t getMeanSideMeasure (const Mesh &m)

Return average measure (length or area) of sides of given mesh.

- void setNodeCodes (Mesh &m, const string &exp, int code, size_t dof=1)

Assign a given code to all nodes satisfying a boolean expression using node coordinates.

- void setBoundaryNodeCodes (Mesh &m, const string &exp, int code, size_t dof=1)

Assign a given code to all nodes on boundary that satisfy a boolean expression using node coordinates.

- void setSideCodes (Mesh &m, const string &exp, int code, size_t dof=1)

Assign a given code to all sides satisfying a boolean expression using node coordinates.

- void setBoundarySideCodes (Mesh &m, const string &exp, int code, size_t dof=1)

Assign a given code to all sides on boundary that satisfy a boolean expression using node coordinates.

- void setElementCodes (Mesh &m, const string &exp, int code)

Assign a given code to all elements satisfying a boolean expression using node coordinates.

- int NodeInElement (const Node *nd, const Element *el)

Say if a given node belongs to a given element.

- int NodeInSide (const Node *nd, const Side *sd)

Say if a given node belongs to a given side.

- int SideInElement (const Side *sd, const Element *el)

Say if a given side belongs to a given element.

- ostream & operator<< (ostream &s, const Node &nd)

Output node data.

- ostream & operator<< (ostream &s, const Side &sd)

Output side data.

5.3.1 Detailed Description

Gathers mesh related classes.

5.3.2 Macro Definition Documentation

#define GRAPH_MEMORY 1000000

Memory necessary to store matrix graph.

This value is necessary only if nodes are to be renumbered.

#define MAX_NB_SUBMESHES 500

Maximum number of mesh partitions.

Useful if mesh is to be partitioned by class Partition

#define MAX_NB_HOLES 10

Maximum number of holes.

Used only if class Domain is invoked

#define MAX_NB_LINES 200

Maximum number of lines.

Used only if class Domain is invoked

#define MAX_NB_LINE_NODES 1000

Maximum number of nodes on each line.

Used only if class Domain is invoked

#define MAX_NB_CONT_LINES 50

Maximum number of lines on each contour.

Used only if class Domain is invoked

#define MAX_NB_SUB_DOMAINS 5

Maximum number of subdomains.

Used only if class Domain is invoked

#define TheNode (*theNode)

A macro that gives the instance pointed by *theNode*

#define TheElement (*theElement)

A macro that gives the instance pointed by *theElement*

#define TheSide (*theSide)

A macro that gives the instance pointed by *theSide*

#define TheEdge (*theEdge)

A macro that gives the instance pointed by *theEdge*

```
#define MeshElements( mesh ) for ((mesh).topElement();  
(theElement=(mesh).getElement());)
```

A macro to loop on mesh elements *mesh*: Instance of Mesh

Note

: Each iteration updates the pointer *theElement* to current Element

```
#define MeshActiveElements( mesh ) for ((mesh).topElement();  
(theElement=(mesh).getActiveElement());)
```

A macro to loop on mesh active elements *mesh* : Instance of Mesh

Note

: Each iteration updates the pointer *theElement* to current Element
: This macro is necessary only if adaptive meshing is used

```
#define MeshNodeLoop( mesh, node ) for ((mesh).topNode();  
((node)=(mesh).getNode());)
```

A macro to loop on mesh nodes *mesh*: Instance of Mesh *node*: Pointer to pointed node

```
#define MeshNodes( mesh ) for ((mesh).topNode(); (theNode=(mesh).getNode());)
```

A macro to loop on mesh nodes *mesh* : Instance of Mesh

Note

: Each iteration updates the pointer *theNode* to current Node

```
#define MeshBoundaryNodes( mesh ) for ((mesh).topBoundaryNode();  
(theNode=(mesh).getBoundaryNode());)
```

A macro to loop on mesh nodes *mesh*: Instance of Mesh

Note

: Each iteration updates the pointer *theNode* to current Node

```
#define MeshSides( mesh ) for ((mesh).topSide(); (theSide=(mesh).getSide());)
```

A macro to loop on mesh sides *mesh*: Instance of Mesh

Note

: Each iteration updates the pointer *theSide* to current Side

```
#define MeshSideSet( sl ) for ((sl).top(); (theSide=(sl).get());)
```

A macro to loop on a subset of mesh sides *sl*: Instance of SideList class

Note

: Each iteration updates the pointer *theSide* to current Side


```
#define MeshBoundarySides( mesh ) for ((mesh).topBoundarySide();  
(theSide=(mesh).getBoundarySide());)
```

A macro to loop on mesh boundary sides mesh: Instance of Mesh

Notes:

- List of boundary sides must have been previously created by using class SideList
- Each iteration updates the pointer theSide to current Side

```
#define MeshEdges( mesh ) for ((mesh).topEdge(); (theEdge=(mesh).getEdge());)
```

A macro to loop on mesh edges mesh: Instance of Mesh

Note

: Each iteration updates the pointer theEdge to current Edge

```
#define theNodeLabel theNode->n()
```

A macro that returns node label in a loop using macro *MeshNodes*

5.3.3 Function Documentation

Figure operator&& (const Figure & f1, const Figure & f2)

Function to define a [Figure](#) instance as the intersection of two [Figure](#) instances.

Function to define a [Figure](#) instance as the union of two [Figure](#) instances.

Parameters

in	f1	First Figure instance
in	f2	Second Figure instance

Returns

Updated resulting [Figure](#) instance

Figure operator- (const Figure & f1, const Figure & f2)

Function to define a [Figure](#) instance as the set subtraction of two [Figure](#) instances.

Parameters

in	f1	First Figure instance to subtract from
in	f2	Second Figure instance to subtract

Returns

Updated resulting [Figure](#) instance

void DeformMesh (Mesh & mesh, const Vect< real_t > & u, real_t a = 1)

Calculate deformed mesh using a displacement field.

Parameters

in,out	<i>mesh</i>	Mesh instance. On output, node coordinates are modified to take into account the displacement
in	<i>u</i>	Displacement field at nodes
in	<i>a</i>	Amplification factor [Default: 1]. The displacement is multiplied by <i>a</i> before to be added to node coordinates

```
void MeshToMesh ( Mesh & m1, Mesh & m2, const Vect< real_t > & u1, Vect< real_t > & u2, size_t nx, size_t ny = 0, size_t nz = 0, size_t dof = 1 )
```

Function to redefine a vector defined on a mesh to a new mesh.

The program interpolates (piecewise linear) first the vector on a finer structured grid. Then the values on the new mesh nodes are computed.

Remarks

For efficiency the number of grid cells must be large enough so that interpolation provides efficient accuracy

Parameters

in	<i>m1</i>	Reference to the first mesh instance
out	<i>m2</i>	Reference to the second mesh instance
in	<i>u1</i>	Input vector of nodal values defined on first mesh
out	<i>u2</i>	Output vector of nodal values defined on second mesh
in	<i>nx</i>	Number of cells in the x-direction in the fine structured grid
in	<i>ny</i>	Number of cells in the y-direction in the fine structured grid The default value of <i>ny</i> is 0, i.e. a 1-D grid
in	<i>nz</i>	Number of cells in the z-direction in the fine structured grid The default value of <i>nz</i> is 0, i.e. a 1-D or 2-D grid
in	<i>dof</i>	Label of degree of freedom of vector <i>u</i> . Only this dof is considered. [Default: 1]

Note

The input vector *u1* is a one degree of freedom per node vector, i.e. its size must be equal (or greater than) the total number of nodes of mesh *m1*. The size of vector *u2* is deduced from the mesh *m2*

```
void MeshToMesh ( Mesh & m1, Mesh & m2, const Vect< real_t > & u1, Vect< real_t > & u2, const Point< real_t > & xmin, const Point< real_t > & xmax, size_t nx, size_t ny, size_t nz, size_t dof = 1 )
```

Function to redefine a vector defined on a mesh to a new mesh.

The program interpolates (piecewise linear) first the vector on a finer structured grid. Then the values on the new mesh nodes are computed. In this function the grid rectangle is defined so that this one can cover only a submesh of *m1*.

Remarks

For efficiency the number of grid cells must be large enough so that interpolation provides efficient accuracy

Parameters

in	<i>m1</i>	Reference to the first mesh instance
out	<i>m2</i>	Reference to the second mesh instance
in	<i>u1</i>	Input vector of nodal values defined on first mesh
out	<i>u2</i>	Output vector of nodal values defined on second mesh
in	<i>xmin</i>	Point instance containing minimal coordinates of the rectangle that defines the grid
in	<i>xmax</i>	Point instance containing maximal coordinates of the rectangle that defines the grid
in	<i>nx</i>	Number of cells in the x-direction in the fine structured grid
in	<i>ny</i>	Number of cells in the y-direction in the fine structured grid The default value of <i>ny</i> is 0, i.e. a 1-D grid
in	<i>nz</i>	Number of cells in the z-direction in the fine structured grid The default value of <i>nz</i> is 0, i.e. a 1-D or 2-D grid
in	<i>dof</i>	Label of degree of freedom of vector <i>u</i> . Only this <i>dof</i> is considered. [Default: 1]

Note

The input vector *u1* is a one degree of freedom per node vector, i.e. its size must be equal (or greater than) the total number of nodes of mesh *m1*. The size of vector *u2* is deduced from the mesh *m2*

real_t getMaxSize (const Mesh & m)

Return maximal size of element edges for given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
----	----------	----------------------------

real_t getMinSize (const Mesh & m)

Return minimal size of element edges for given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
----	----------	----------------------------

real_t getMinElementMeasure (const Mesh & m)

Return minimal measure (length, area or volume) of elements of given mesh.

Return maximal measure (length, area or volume) of elements of given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
----	----------	----------------------------

real_t getMinSideMeasure (const Mesh & m)

Return minimal measure (length or area) of sides of given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
-----------	----------	----------------------------

Note

Use this function only if sides are present in the mesh and for 2-D meshes

real_t getMaxSideMeasure (const Mesh & m)

Return maximal measure (length or area) of sides of given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
-----------	----------	----------------------------

Note

Use this function only if sides are present in the mesh and for 2-D meshes

real_t getMeanElementMeasure (const Mesh & m)

Return average measure (length, area or volume) of elements of given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
-----------	----------	----------------------------

real_t getMeanSideMeasure (const Mesh & m)

Return average measure (length or area) of sides of given mesh.

Parameters

in	<i>m</i>	Reference to mesh instance
-----------	----------	----------------------------

Note

Use this function only if sides are present in the mesh and for 2-D meshes

void setNodeCodes (Mesh & m, const string & exp, int code, size_t dof = 1)

Assign a given code to all nodes satisfying a boolean expression using node coordinates.

Parameters

in	<i>m</i>	Reference to mesh instance
in	<i>exp</i>	Regular expression using x, y, and z coordinates of nodes, according to <code>fparser</code> parser
in	<i>code</i>	Code to assign
in	<i>dof</i>	Degree of freedom for which code is assigned [Default: 1]

void setBoundaryNodeCodes (Mesh & m, const string & exp, int code, size_t dof = 1)

Assign a given code to all nodes on boundary that satisfy a boolean expression using node coordinates.

Parameters

in	<i>m</i>	Reference to mesh instance
in	<i>exp</i>	Regular expression using x, y, and z coordinates of nodes, according to <code>fparser</code> parser
in	<i>code</i>	Code to assign
in	<i>dof</i>	Degree of freedom for which code is assigned [Default: 1]

void setSideCodes (Mesh & m, const string & exp, int code, size_t dof = 1)

Assign a given code to all sides satisfying a boolean expression using node coordinates.

Parameters

in	<i>m</i>	Reference to mesh instance
in	<i>exp</i>	Regular expression using x, y, and z coordinates of side nodes, according to <code>fparser</code> parser
in	<i>code</i>	Code to assign
in	<i>dof</i>	Degree of freedom for which code is assigned [Default: 1]

void setBoundarySideCodes (Mesh & m, const string & exp, int code, size_t dof = 1)

Assign a given code to all sides on boundary that satisfy a boolean expression using node coordinates.

Parameters

in	<i>m</i>	Reference to mesh instance
in	<i>exp</i>	Regular expression using x, y, and z coordinates of side nodes, according to <code>fparser</code> parser
in	<i>code</i>	Code to assign
in	<i>dof</i>	Degree of freedom for which code is assigned [Default: 1]

void setElementCodes (Mesh & m, const string & exp, int code)

Assign a given code to all elements satisfying a boolean expression using node coordinates.

Parameters

in	<i>m</i>	Reference to mesh instance
in	<i>exp</i>	Regular expression using x, y, and z coordinates of element nodes, according to <code>fparser</code> parser
in	<i>code</i>	Code to assign

int NodeInElement (const Node * nd, const Element * el)

Say if a given node belongs to a given element.

Parameters

in	<i>nd</i>	Pointer to Node
in	<i>el</i>	Pointer to Element

Returns

Local label of the node if this one is found, 0 if not.

int NodeInSide (const Node * nd, const Side * sd)

Say if a given node belongs to a given side.

Parameters

<code>in</code>	<code>nd</code>	Pointer to Node
<code>in</code>	<code>sd</code>	Pointer to Side

Returns

Local label of the node if this one is found, 0 if not.

int SideInElement (const Side * sd, const Element * el)

Say if a given side belongs to a given element.

Parameters

<code>in</code>	<code>sd</code>	Pointer to Side
<code>in</code>	<code>el</code>	Pointer to Element

Returns

Local label of the side if this one is found, 0 if not.

5.4 Fluid Dynamics

Gathers Fluid Dynamics equation classes.

Classes

- class [Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ >](#)
Abstract class for Fluid Dynamics [Equation](#) classes.
- class [NSP2DQ41](#)
Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using Q_1/P_0 element and a penalty formulation for the incompressibility condition.
- class [TINS2DT3B](#)
Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

5.4.1 Detailed Description

Gathers Fluid Dynamics equation classes.

5.5 General Purpose Equations

Gathers equation related classes.

Classes

- class [AbsEqua< T_ >](#)
Mother abstract class to describe equation.
- class [Equation< T_, NEN_, NEE_, NSN_, NSE_ >](#)
Abstract class for all equation classes.
- class [Estimator](#)
To calculate an a posteriori estimator of the solution.

Functions

- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalVect< T_ , N_ > &be, Vect< T_ > &b)
Assemble local vector into global vector.
- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalMatrix< T_ , N_ , N_ > &ae, Vect< T_ > &b)
Assemble diagonal local vector into global vector.
- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalMatrix< T_ , N_ , N_ > &ae, Matrix< T_ > *A)
Assemble local matrix into global matrix.
- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalMatrix< T_ , N_ , N_ > &ae, SkMatrix< T_ > &A)
Assemble local matrix into global skyline matrix.
- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalMatrix< T_ , N_ , N_ > &ae, SkSMatrix< T_ > &A)
Assemble local matrix into global symmetric skyline matrix.
- template<class T_ , size_t N_ , class E_ >
void [element_assembly](#) (const E_ &e, const LocalMatrix< T_ , N_ , N_ > &ae, SpMatrix< T_ > &A)
Assemble local matrix into global sparse matrix.
- template<class T_ , size_t N_ >
void [side_assembly](#) (const Element &e, const LocalMatrix< T_ , N_ , N_ > &ae, SpMatrix< T_ > &A)
Side assembly of local matrix into global matrix (as instance of class [SpMatrix](#)).
- template<class T_ , size_t N_ >
void [side_assembly](#) (const Element &e, const LocalMatrix< T_ , N_ , N_ > &ae, SkSMatrix< T_ > &A)
Side assembly of local matrix into global matrix (as instance of class [SkSMatrix](#)).
- template<class T_ , size_t N_ >
void [side_assembly](#) (const Element &e, const LocalMatrix< T_ , N_ , N_ > &ae, SkMatrix< T_ > &A)
Side assembly of local matrix into global matrix (as instance of class [SkMatrix](#)).
- template<class T_ , size_t N_ >
void [side_assembly](#) (const Element &e, const LocalVect< T_ , N_ > &be, Vect< T_ > &b)

Side assembly of local vector into global vector.

- ostream & [operator<<](#) (ostream &s, const Estimator &r)

Output estimator vector in output stream.

5.5.1 Detailed Description

Gathers equation related classes.

5.5.2 Function Documentation

void element_assembly (const E_ & e, const LocalVect< T_, N_ > & be, Vect< T_ > & b)

Assemble local vector into global vector.

Parameters

in	e	Reference to local entity (Element or Side)
in	be	Local vector
in,out	b	Global vector

void element_assembly (const E_ & e, const LocalMatrix< T_, N_, N_ > & ae, Vect< T_ > & b)

Assemble diagonal local vector into global vector.

Parameters

in	e	Reference to local entity (Element or Side)
in	ae	Local matrix
in,out	b	Global vector

void element_assembly (const E_ & e, const LocalMatrix< T_, N_, N_ > & ae, Matrix< T_ > * A)

Assemble local matrix into global matrix.

This function is to be called with an abstract pointer to matrix (class [Matrix](#))

Parameters

in	e	Reference to local entity (Element or Side)
in	ae	Local matrix
in,out	A	Pointer to global matrix

void element_assembly (const E_ & e, const LocalMatrix< T_, N_, N_ > & ae, SkMatrix< T_ > & A)

Assemble local matrix into global skyline matrix.

Parameters

in	e	Reference to local entity (Element or Side)
in	ae	Local matrix

in,out	A	Global matrix
--------	---	---------------

void element_assembly (const E_ & e, const LocalMatrix< T_, N_, N_ > & ae, SkSMatrix< T_ > & A)

Assemble local matrix into global symmetric skyline matrix.

Parameters

in	e	Reference to local entity (Element or Side)
in	ae	Local matrix
in,out	A	Global matrix

void element_assembly (const E_ & e, const LocalMatrix< T_, N_, N_ > & ae, SpMatrix< T_ > & A)

Assemble local matrix into global sparse matrix.

Parameters

in	e	Reference to local entity (Element or Side)
in	ae	Local matrix
in,out	A	Global matrix

void side_assembly (const Element & e, const LocalMatrix< T_, N_, N_ > & ae, SpMatrix< T_ > & A)

Side assembly of local matrix into global matrix (as instance of class [SpMatrix](#)).

Parameters

in	e	Reference to local Element
in	ae	Local matrix
in,out	A	Global matrix

void side_assembly (const Element & e, const LocalMatrix< T_, N_, N_ > & ae, SkSMatrix< T_ > & A)

Side assembly of local matrix into global matrix (as instance of class [SkSMatrix](#)).

Parameters

in	e	Reference to local Element
in	ae	Local matrix
in,out	A	Global matrix

void side_assembly (const Element & e, const LocalMatrix< T_, N_, N_ > & ae, SkMatrix< T_ > & A)

[Side](#) assembly of local matrix into global matrix (as instance of class [SkMatrix](#)).

Parameters

in	<i>e</i>	Reference to local Element
in	<i>ae</i>	Local matrix
in,out	<i>A</i>	Global matrix

void side_assembly (const [Element](#) & e, const LocalVect< T₋, N₋ > & be, Vect< T₋ > & b)

Side assembly of local vector into global vector.

Parameters

in	<i>e</i>	Reference to local Element
in	<i>be</i>	Local vector
in,out	<i>b</i>	Global vector

5.6 Global Variables

Gathers all global variables in the library.

Variables

- Node * [theNode](#)
A pointer to [Node](#).
- Element * [theElement](#)
A pointer to [Element](#).
- Side * [theSide](#)
A pointer to [Side](#).
- Edge * [theEdge](#)
A pointer to [Edge](#).
- unsigned int [theStep](#)
Time step counter.
- unsigned int [theliteration](#)
Iteration counter.
- unsigned int [NbTimeSteps](#)
Number of time steps.
- unsigned int [MaxNbIterations](#)
Maximal number of iterations.
- unsigned int [Verbosity](#)
Parameter for verbosity of message outputting.
- double [theTimeStep](#)
Time step label.
- double [theTime](#)
Time value.
- double [theFinalTime](#)
Final time value.
- double [theTolerance](#)
Tolerance value for convergence.
- double [theDiscrepancy](#)
Value of discrepancy for an iterative procedure Its default value is 1. 0.
- bool [Converged](#)
Boolean variable to say if an iterative procedure has converged.
- bool [InitPetsc](#)

5.6.1 Detailed Description

Gathers all global variables in the library.

5.6.2 Variable Documentation

Node* [theNode](#)

A pointer to [Node](#).

Useful for loops on nodes

Element* theElement

A pointer to [Element](#).

Useful for loops on elements

Side* theSide

A pointer to [Side](#).

Useful for loops on sides

Edge* theEdge

A pointer to [Edge](#).

Useful for loops on edges

unsigned int theStep

Time step counter.

This counter must be initialized by the user if the macro `timeLoop` is not used

Remarks

May be used in conjunction with the macro `TimeLoop`. In this case, it has to be initialized before.
Its default value is 1

unsigned int theIteration

Iteration counter.

This counter must be initialized by the user

Remarks

May be used in conjunction with the macro `IterationLoop`. Its default value is 1

unsigned int NbTimeSteps

Number of time steps.

Remarks

May be used in conjunction with the macro `TimeLoop`.

unsigned int MaxNbIterations

Maximal number of iterations.

Remarks

May be used in conjunction with the macro `IterationLoop`. Its default value is 1000

unsigned int Verbosity

Parameter for verbosity of message outputting.

Its default value is 1

double theTimeStep

Time step label.

Remarks

May be used in conjunction with the macro TimeLoop. In this case, it has to be initialized before

double theTime

Time value.

Remarks

May be used in conjunction with the macro TimeLoop. Its default value is 0.0

double theFinalTime

Final time value.

Remarks

May be used in conjunction with the macro TimeLoop. In this case, it has to be initialized before

double theTolerance

Tolerance value for convergence.

Remarks

May be used within an iterative procedure. Its default value is 1.e-8

bool Converged

Boolean variable to say if an iterative procedure has converged.

Its default value is false

bool InitPetsc

Boolean to say if PETSc use was initialized. Useful only if PETSc is used

5.7 Heat Transfer

Gathers Heat Transfer equation classes.

Classes

- class [DC1DL2](#)
Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.
- class [DC2DT3](#)
Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.
- class [DC2DT6](#)
Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.
- class [DC3DAT3](#)
Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.
- class [DC3DT4](#)
Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.
- class [Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ >](#)
Abstract class for Heat transfer Finite Element classes.
- class [PhaseChange](#)
This class enables defining phase change laws for a given material.

5.7.1 Detailed Description

Gathers Heat Transfer equation classes.

5.8 Input/Output

Gathers Input/Output utility classes.

Classes

- class [IOField](#)
Enables working with files in the XML Format.
- class [IPF](#)
To read project parameters from a file in [IPF](#) format.
- class [Prescription](#)
To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

Macros

- `#define` [MAX_NB_PAR](#) 50
Maximum number of parameters.
- `#define` [MAX_ARRAY_SIZE](#) 100
Maximum array size.
- `#define` [MAX_INPUT_STRING_LENGTH](#) 100
Maximum string length.
- `#define` [FILENAME_LENGTH](#) 150
Length of a string defining a file name.

5.8.1 Detailed Description

Gathers Input/Output utility classes.

5.8.2 Macro Definition Documentation

`#define` [MAX_NB_PAR](#) 50

Maximum number of parameters.
Used in class [IPF](#)

`#define` [MAX_ARRAY_SIZE](#) 100

Maximum array size.
Used in class [IPF](#)

`#define` [MAX_INPUT_STRING_LENGTH](#) 100

Maximum string length.
Used in class [IPF](#)

5.9 Interface Problems

Gathers classes and functions for general interface problems, including image processing.

Classes

- class [FastMarching2D](#)
To run a Fast Marching Method on 2-D structured uniform grids.
- class [FMMSolver](#)
The Fast Marching Method solver.

5.9.1 Detailed Description

Gathers classes and functions for general interface problems, including image processing.

5.10 Laplace equation

Gathers equation classes for the Laplace equation.

Classes

- class [Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ >](#)
Abstract class for classes about the Laplace equation.
- class [Laplace1DL2](#)
To build element equation for a 1-D elliptic equation using the 2-Node line element (P_1).
- class [Laplace1DL3](#)
To build element equation for the 1-D elliptic equation using the 3-Node line (P_2).
- class [Laplace2DFVT](#)
To build and solve the Laplace equation using a standard Finite Volume method.
- class [Laplace2DMHRT0](#)
To build element equation for the 2-D elliptic equation using the Mixed Hybrid finite element at lowest degree (Raviart-Thomas RT_0).
- class [Laplace2DT3](#)
To build element equation for the Laplace equation using the 2-D triangle element (P_1).
- class [SteklovPoincare2DBE](#)
Solver of the Steklov Poincare problem in 2-D geometries using piecewise constant boundary elemen.

5.10.1 Detailed Description

Gathers equation classes for the Laplace equation.

5.11 OFELI

Namespaces

- [OFELI](#)

A namespace to group all library classes, functions, ...

5.11.1 Detailed Description

5.12 Physical properties of media

A module that contains classes for physical properties of materials and media.

Classes

- class [Material](#)

To treat material data. This class enables reading material data in material data files. It also returns these informations by means of its members.

5.12.1 Detailed Description

A module that contains classes for physical properties of materials and media.

5.13 Shape Function

Gathers shape function classes.

Classes

- class [FEShape](#)
Parent class from which inherit all finite element shape classes.
- class [triangle](#)
Defines a triangle. The reference element is the rectangle triangle with two unit edges.
- class [Hexa8](#)
Defines a three-dimensional 8-node hexahedral finite element using Q_1 -isoparametric interpolation.
- class [Line2](#)
To describe a 2-Node planar line finite element.
- class [Line2H](#)
To describe a 2-Node Hermite planar line finite element.
- class [Line3](#)
To describe a 3-Node quadratic planar line finite element.
- class [Penta6](#)
Defines a 6-node pentahedral finite element using P_1 interpolation in local coordinates $(s.x, s.y)$ and Q_1 isoparametric interpolation in local coordinates $(s.x, s.z)$ and $(s.y, s.z)$.
- class [Quad4](#)
Defines a 4-node quadrilateral finite element using Q_1 isoparametric interpolation.
- class [Tetra4](#)
Defines a three-dimensional 4-node tetrahedral finite element using P_1 interpolation.
- class [Triang3](#)
Defines a 3-Node (P_1) triangle.
- class [Triang6S](#)
Defines a 6-Node straight triangular finite element using P_2 interpolation.

5.13.1 Detailed Description

Gathers shape function classes.

5.14 Solid Mechanics

Gathers Solid Mechanics finite element equations related classes.

Classes

- class [Bar2DL2](#)
To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node.
- class [Beam3DL2](#)
To build element equations for 3-D beam equations using 2-node lines.
- class [Elas2DQ4](#)
To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.
- class [Elas2DT3](#)
To build element equations for 2-D linearized elasticity using 3-node triangles.
- class [Elas3DH8](#)
To build element equations for 3-D linearized elasticity using 8-node hexahedra.
- class [Elas3DT4](#)
To build element equations for 3-D linearized elasticity using 4-node tetrahedra.
- class [Equa.Solid< T_, NEN_, NEE_, NSN_, NSE_ >](#)
Abstract class for Solid Mechanics Finite Element classes.

5.14.1 Detailed Description

Gathers Solid Mechanics finite element equations related classes.

5.15 Solver

Gathers Solver functions.

Files

- file [OptimAux.h](#)
File that contains auxiliary functions for optimization.
- file [OptimSA.h](#)
Function to solve an optimization problem using the Simulated Annealing method.
- file [OptimTN.h](#)
Function to solve an optimization problem using the Truncated Newton method.

Classes

- class [Reconstruction](#)
To perform various reconstruction operations.
- class [Eigen](#)
Class to find the smallest eigenvalues and corresponding eigenvectors in a generalized eigenproblem using the Bathe subspace iteration method.
- class [Iter< T_ >](#)
Class to drive an iterative process.
- class [Prec< T_ >](#)
To set a preconditioner.
- class [Precond< T_ >](#)
An abstract class from which derive all preconditioning classes.

Macros

- `#define` [MAX_NB_EQUATIONS](#) 5
Maximum number of equations.
- `#define` [MAX_NB_INPUT_FIELDS](#) 3
Maximum number of fields for an equation.
- `#define` [MAX_NB_MESHES](#) 10
Maximum number of meshes.
- `#define` [TIME_LOOP](#)(ts, t, ft, n)
A macro to loop on time steps to integrate on time ts : Time step t : Initial time value updated at each time step ft : Final time value n : Time step index.
- `#define` [TimeLoop](#)
A macro to loop on time steps to integrate on time.
- `#define` [IterationLoop](#) while (++theliteration<MaxNbIterations && Converged==false)
A macro to loop on iterations for an iterative procedure.

Functions

- `ostream & operator<<` (`ostream &s`, `const Muscl3DT &m`)
Output mesh data as calculated in class [Muscl3DT](#).
- `template<class T_ , class M_ , class P_ >`
`int BiCG` (`const M_ &A`, `const P_ &P`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `int max_it`, `double &toler`, `int verbose`)
Biconjugate gradient solver function.
- `template<class T_ , class M_ , class P_ >`
`int BiCGStab` (`const M_ &A`, `const P_ &P`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `int max_it`, `double &toler`, `int verbose`)
Biconjugate gradient stabilized solver function.
- `template<class T_ , class M_ , class P_ >`
`int CG` (`const M_ &A`, `const P_ &P`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `int max_it`, `double &toler`, `int verbose`)
Conjugate gradient solver function.
- `template<class T_ , class M_ , class P_ >`
`int CGS` (`const M_ &A`, `const P_ &P`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `int max_it`, `real_t &toler`, `int verbose`)
Conjugate Gradient Squared solver function.
- `template<class T_ , class M_ , class P_ >`
`int GMRes` (`const M_ &A`, `const P_ &P`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `size_t m`, `int max_it`, `double &toler`, `int verbose`)
GMRes solver function.
- `template<class T_ , class M_ >`
`int GS` (`const M_ &A`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `real_t omega`, `int max_it`, `real_t &toler`, `int verbose`)
Gauss-Seidel solver function.
- `template<class T_ , class M_ >`
`int Jacobi` (`const M_ &A`, `const Vect< T_ > &b`, `Vect< T_ > &x`, `real_t omega`, `int max_it`, `real_t toler`, `int verbose`)
Jacobi solver function.
- `void BCAsConstraint` (`const Mesh &m`, `const Vect< real_t > &bc`, `Vect< real_t > &up`, `Vect< real_t > &low`)
*To impose Dirichlet boundary conditions in an optimization problem. If such conditions are to present, this function has to be invoked by giving on input `bc(i)` as the value to impose for the *i*-th optimization variable.*
- `template<class OPT_ >`
`int OptimSA` (`OPT_ &theOpt`, `Vect< real_t > &x`, `real_t &rt`, `real_t &eps`, `int &ns`, `int &nt`, `int &neps`, `int &maxevl`, `Vect< real_t > &lb`, `Vect< real_t > &ub`, `Vect< real_t > &c`, `int &msg_lvl`, `int &seed1`, `int &seed2`, `real_t &t`, `Vect< real_t > &vm`, `Vect< real_t > &xopt`, `real_t &fopt`, `int &nacc`, `int &nfcnev`, `int &nobds`)
Simulated annealing optimization solver.
- `template<class OPT_ >`
`int OptimTN` (`OPT_ &theOpt`, `Vect< real_t > &x`, `Vect< real_t > &low`, `Vect< real_t > &up`, `Vect< int > &pivot`, `int max_it`, `real_t toler`, `int msg_lvl`)
Truncated Newton optimization solver.
- `template<class M_ , class P1_ , class P2_ >`
`int QMR` (`const M_ &A`, `const P1_ &P1`, `const P2_ &P2`, `const Vect< real_t > &b`, `Vect< real_t > &x`, `int max_it`, `real_t &toler`, `int verbose`)
QMR solver function.

- `template<class T_ , class M_ >`
`int Richardson (const M_ &A, const Vect< T_ > &b, Vect< T_ > &x, real_t omega, int max_it, real_t toler, int verbose)`
Richardson solver function.
- `template<class T_ , class MAT_A_ , class MAT_D_ , class MAT_U_ >`
`void Schur (MAT_A_ &A, MAT_U_ &U, MAT_U_ &L, MAT_D_ &D, Vect< T_ > &b, Vect< T_ > &c)`
Solve a linear system of equations with a 2x2-block matrix.
- `template<class T_ , class MAT_A_ , class MAT_D_ , class MAT_U_ >`
`void Schur (Vect< MAT_A_ > &A, Vect< MAT_U_ > &U, Vect< MAT_U_ > &L, MAT_D_ &D, const Vect< Vect< T_ > > &b, const Vect< T_ > &c, Vect< Vect< T_ > > &x, Vect< T_ > &y)`
Solve a linear arrow block system by block factorization.
- `template<class T_ , class M_ >`
`int SSOR (const M_ &A, const Vect< T_ > &b, Vect< T_ > &x, int max_it, double toler, int verbose)`
SSOR solver function.

5.15.1 Detailed Description

Gathers Solver functions.

5.15.2 Macro Definition Documentation

#define MAX_NB_EQUATIONS 5

Maximum number of equations.
 Useful for coupled problems

#define MAX_NB_INPUT_FIELDS 3

Maximum number of fields for an equation.
 Useful for coupled problems

#define MAX_NB_MESHES 10

Maximum number of meshes.
 Useful for coupled problems

#define TimeLoop

Value:

```
NbTimeSteps = theFinalTime/theTimeStep; \
  for (theTime=theTimeStep, theStep=1; theTime<
    theFinalTime+0.001*theTimeStep; theTime+=
    theTimeStep, ++theStep)
```

A macro to loop on time steps to integrate on time.

It uses the following global variables defined in [OFELI](#): theStep, theTime, theTimeStep, theFinalTime

#define IterationLoop while (++theIteration<MaxNbIterations && Converged==false)

A macro to loop on iterations for an iterative procedure.

It uses the following global variables defined in [OFELI](#): theIteration, MaxNbIterations, Converged

5.15.3 Function Documentation

int BiCG (const M_ & A, const P_ & P, const Vect< T_ > & b, Vect< T_ > & x, int max_it, double & toler, int verbose)

Biconjugate gradient solver function.

Parameters

in	A	Problem matrix (Instance of abstract class M_).
in	P	Preconditioner (Instance of abstract class P_).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max_it	Maximum number of iterations.
	$toler$	[in,out] Tolerance for convergence (measured in relative weighted 2--Norm) in input, effective discrepancy in output.
	$verbose$	[in] Information output parameter <ul style="list-style-type: none"> • 0: No output • 1: Output iteration information, • 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Parameters

$\langle T_ \rangle$	Data type (double, float, complex<double>, ...)
$\langle M_ \rangle$	Matrix storage class
$\langle P_ \rangle$	Preconditioner class

int BiCGStab (const $M_$ & A, const $P_$ & P, const **Vect< $T_$ > & b, **Vect**< $T_$ > & x, int max.it, double & toler, int verbose)**

Biconjugate gradient stabilized solver function.

Parameters

in	A	Problem matrix (Instance of abstract class M_).
in	P	Preconditioner (Instance of abstract class P_).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max_it	Maximum number of iterations.
in,out	$toler$	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.
in	$verbose$	Information output parameter <ul style="list-style-type: none"> • 0: No output • 1: Output iteration information, • 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Parameters

$\langle T_ \rangle$	Data type (double, float, complex<double>, ...)
$\langle M_ \rangle$	Matrix storage class
$\langle P_ \rangle$	Preconditioner class

int CG (const $M_$ & A, const $P_$ & P, const Vect< $T_$ > & b, Vect< $T_$ > & x, int max_it, double & toler, int verbose)

Conjugate gradient solver function.

Parameters

in	A	Problem matrix (Instance of abstract class $M_$).
in	P	Preconditioner (Instance of abstract class $P_$).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	max_it	Maximum number of iterations.
in	$toler$	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.
in	$verbose$	Information output parameter <ul style="list-style-type: none"> • 0: No output • 1: Output iteration information, • 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Parameters

$\langle T_ \rangle$	Data type (double, float, complex<double>, ...)
$\langle M_ \rangle$	Matrix storage class
$\langle P_ \rangle$	Preconditioner class

int CGS (const $M_$ & A, const $P_$ & P, const Vect< $T_$ > & b, Vect< $T_$ > & x, int max_it, real_t & toler, int verbose)

Conjugate Gradient Squared solver function.

Parameters

in	A	Problem matrix (Instance of abstract class $M_$).
in	P	Preconditioner (Instance of abstract class $P_$).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).

<code>in</code>	<code>max_it</code>	Maximum number of iterations.
<code>in,out</code>	<code>toler</code>	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.
<code>in</code>	<code>verbose</code>	Information output parameter <ul style="list-style-type: none"> • 0: No output • 1: Output iteration information, • 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Parameters

<code><T_></code>	Data type (real_t, float, complex<real_t>, ...)
<code><M_></code>	Matrix storage class
<code><P_></code>	Preconditioner class

int GMRes (const M_ & A, const P_ & P, const Vect< T_ > & b, Vect< T_ > & x, size_t m, int max_it, double & toler, int verbose)

GMRes solver function.

Parameters

<code>in</code>	<code>A</code>	Problem matrix (Instance of abstract class M_).
<code>in</code>	<code>P</code>	Preconditioner (Instance of abstract class P_).
<code>in</code>	<code>b</code>	Right-hand side vector (class Vect)
<code>in,out</code>	<code>x</code>	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
<code>in</code>	<code>m</code>	Number of subspaces to generate for iterations.
<code>in</code>	<code>max_it</code>	Maximum number of iterations.
<code>in,out</code>	<code>toler</code>	Tolerance for convergence (measured in relative weighted 2-Norm) on input. Actual convergence tolerance on output
<code>in</code>	<code>verbose</code>	Information output parameter (0: No output, 1: Output iteration information, 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
<code><M_></code>	Matrix storage class

$\langle P_ \rangle$	Preconditioner class
-----------------------	----------------------

int GS (const M_ & A, const Vect< T_ > & b, Vect< T_ > & x, real_t omega, int max_it, real_t & toler, int verbose)

Gauss-Seidel solver function.

Parameters

in	<i>A</i>	Problem matrix (Instance of abstract class M_).
in	<i>b</i>	Right-hand side vector (class Vect)
in,out	<i>x</i>	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	<i>omega</i>	Relaxation parameter.
in	<i>max_it</i>	Maximum number of iterations.
in	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.
in	<i>verbose</i>	Information output parameter <ul style="list-style-type: none"> • 0: No output • 1: Output iteration information • 2 and greater: Output iteration information and solution at each iteration.

Returns

Number of performed iterations

Template Parameters

$\langle T_ \rangle$	Data type (real_t, float, complex<real_t>, ...)
$\langle M_ \rangle$	Matrix storage class

int Jacobi (const M_ & A, const Vect< T_ > & b, Vect< T_ > & x, real_t omega, int max_it, real_t toler, int verbose)

Jacobi solver function.

Parameters

in	<i>A</i>	Problem matrix (Instance of abstract class M_).
in	<i>b</i>	Right-hand side vector (class Vect)
in,out	<i>x</i>	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	<i>omega</i>	Relaxation parameter.
in	<i>max_it</i>	Maximum number of iterations.
in,out	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.

<code>in</code>	<code>verbose</code>	Information output parameter (0: No output, 1: Output iteration information, 2 and greater: Output iteration information and solution at each iteration.
-----------------	----------------------	--

Returns

Number of performed iterations,

Template Parameters

<code><T_></code>	Data type (real_t, float, complex<real_t>, ...)
<code><M_></code>	Matrix storage class

int void BCAsConstraint (const Mesh & m, const Vect< real_t > & bc, Vect< real_t > & up, Vect< real_t > & low)

To impose Dirichlet boundary conditions in an optimization problem. If such conditions are to present, this function has to be invoked by giving on input `bc(i)` as the value to impose for the *i*-th optimization variable.

Parameters

<code>in</code>	<code>m</code>	Mesh instance
<code>in</code>	<code>bc</code>	Vect instance where <code>bc(i)</code> is the value to impose for dof <i>i</i>
<code>out</code>	<code>up</code>	Vect instance that contains on output upper bounds for DOFs
<code>out</code>	<code>low</code>	Vect instance that contains on output lower bounds for DOFs

int OptimSA (OPT_ & theOpt, Vect< real_t > & x, real_t & rt, real_t & eps, int & ns, int & nt, int & neps, int & maxevl, Vect< real_t > & lb, Vect< real_t > & ub, Vect< real_t > & c, int & msg_lvl, int & seed1, int & seed2, real_t & t, Vect< real_t > & vm, Vect< real_t > & xopt, real_t & fopt, int & nacc, int & nfcnev, int & nobds)

Simulated annealing optimization solver.

Simulated annealing is a global optimization method that distinguishes between different local optima. Starting from an initial point, the algorithm takes a step and the function is evaluated. When minimizing a function, any downhill step is accepted and the process repeats from this new point. An uphill step may be accepted. Thus, it can escape from local optima. This uphill decision is made by the Metropolis criteria. As the optimization process proceeds, the length of the stx decline and the algorithm closes in on the global optimum. Since the algorithm makes very few assumptions regarding the function to be optimized, it is quite robust with respect to non-quadratic surfaces. The degree of robustness can be adjusted by the user. In fact, simulated annealing can be used as a local optimizer for difficult functions.

This implementation of simulated annealing was used in "Global Optimization of Statistical Functions with Simulated Annealing," Goffe, Ferrier and Rogers, Journal of Econometrics, vol. 60, no. 1/2, Jan./Feb. 1994, pp. 65-100. Briefly, we found it competitive, if not superior, to multiple restarts of conventional optimization routines for difficult optimization problems.

For more information on this routine, contact its author: Bill Goffe, bgoffe@whale.st.usm.edu

Synopsis: This function implements the continuous simulated annealing global optimization algorithm described in Corana et al.'s article "Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm" in the September 1987 (vol. 13, no. 3, pp. 262-280) issue of the ACM Transactions on Mathematical Software.

A very quick (perhaps too quick) overview of OptimSA:

OptimSA tries to find the global optimum of an N dimensional function. It moves both up and downhill and as the optimization process proceeds, it focuses on the most promising area.

To start, it randomly chooses a trial point within the step length vm of the user selected starting point. The function is evaluated at this trial point and its value is compared to its value at the initial point.

In a maximization problem, all uphill moves are accepted and the algorithm continues from that trial point. Downhill moves may be accepted; the decision is made by the Metropolis criteria. It uses T (temperature) and the size of the downhill move in a probabilistic manner. The smaller t and the size of the downhill move are, the more likely that move will be accepted. If the trial is accepted, the algorithm moves on from that point. If it is rejected, another point is chosen instead for a trial evaluation.

Each element of vm periodically adjusted so that half of all function evaluations in that direction are accepted.

A fall in t is imposed upon the system with the rt variable by $t(i+1) = rt*t(i)$ where i is the i -th iteration. Thus, as t declines, downhill moves are less likely to be accepted and the percentage of rejections rise. Given the scheme for the selection for vm , vm falls. Thus, as t declines, vm falls and OptimSA focuses upon the most promising area for optimization.

The importance of the parameter t :

The parameter t is crucial in using OptimSA successfully. It influences vm , the step length over which the algorithm searches for optima. For a small initial t , the step length may be too small; thus not enough of the function might be evaluated to find the global optima. The user should carefully examine vm in the intermediate output (set `msg.lvl = 1`) to make sure that vm is appropriate. The relationship between the initial temperature and the resulting step length is function dependent.

To determine the starting temperature that is consistent with optimizing a function, it is worthwhile to run a trial run first. Set $rt = 1.5$ and $t = 1.0$. With $rt > 1.0$, the temperature increases and vm rises as well. Then select the T that produces a large enough vm .

For modifications to the algorithm and many details on its use, (particularly for econometric applications) see Goffe, Ferrier and Rogers, "Global Optimization of Statistical Functions with Simulated Annealing," Journal of Econometrics, vol. 60, no. 1/2, Jan./Feb. 1994, pp. 65-100.

For more information, contact

Bill Goffe

Department of Economics and International Business

University of Southern Mississippi

Hattiesburg, MS 39506-5072

(601) 266-4484 (office)

(601) 266-4920 (fax)

bgoffe@whale.st.usm.edu (Internet)

As far as possible, the parameters here have the same name as in the description of the algorithm on pp. 266-8 of Corana et al.

Note: The suggested values generally come from Corana et al. To drastically reduce runtime, see Goffe et al., pp. 90-1 for suggestions on choosing the appropriate rt and nt .

Parameters

in	<i>theOpt</i>	Instance of class <i>OPT_</i> that is implemented by the user and that provides the objective function.
in	<i>x</i>	The starting values for the variables of the function to be optimized.
in	<i>rt</i>	The temperature reduction factor. The value suggested by Corana et al. is .85. See Goffe et al. for more advice.
in	<i>eps</i>	Error tolerance for termination. If the final function values from the last neps temperatures differ from the corresponding value at the current temperature by less than eps and the final function value at the current temperature differs from the current optimal function value by less than eps, execution terminates and the value 0 is returned.
in	<i>ns</i>	Number of cycles. After $ns*n$ function evaluations, each element of <i>vm</i> is adjusted so that approximately half of all function evaluations are accepted. The suggested value is 20.
in	<i>nt</i>	Number of iterations before temperature reduction. After $nt*ns*n$ function evaluations, temperature (<i>t</i>) is changed by the factor <i>rt</i> . Value suggested by Corana et al. is $\max(100, 5*n)$. See Goffe et al. for further advice.
in	<i>neps</i>	Number of final function values used to decide upon termination. See <i>eps</i> . Suggested value is 4
	<i>maxevl</i>	[in] The maximum number of function evaluations. If it is exceeded, the return <i>code</i> =1.
in	<i>lb</i>	The lower bound for the allowable solution variables.
in	<i>ub</i>	The upper bound for the allowable solution variables. If the algorithm chooses $x(i) < lb(i)$ or $x(i) > ub(i)$ $i=1, \dots, n$, a point is from inside is randomly selected. This focuses the algorithm on the region inside <i>ub</i> and <i>lb</i> . Unless the user wishes to concentrate the search to a particular region, <i>ub</i> and <i>lb</i> should be set to very large positive and negative values, respectively. Note that the starting vector <i>x</i> should be inside this region. Also note that <i>lb</i> and <i>ub</i> are fixed in position, while <i>vm</i> is centered on the last accepted trial set of variables that optimizes the function.
	<i>c</i>	[in] Vector that controls the step length adjustment. The suggested value for all elements is 2.
in	<i>msg_lvl</i>	controls printing inside <i>OptimSA</i> . <ul style="list-style-type: none"> • 0 - Nothing printed. • 1 - Function value for the starting value and summary results before each temperature reduction. This includes the optimal function value found so far, the total number of moves (broken up into uphill, downhill, accepted and rejected), the number of out of bounds trials, the number of new optima found at this temperature, the current optimal <i>x</i> and the step length <i>vm</i>. Note that there are $n*ns*nt$ function evaluations before each temperature reduction. Finally, notice is also given upon achieving the termination criteria. • 2 - Each new step length (<i>vm</i>), the current optimal <i>x</i> (<i>xopt</i>) and the current trial <i>x</i> (<i>x</i>). This gives the user some idea about how far <i>x</i> strays from <i>xopt</i> as well as how <i>vm</i> is adapting to the function. • 3 - Each function evaluation, its acceptance or rejection and new optima. For many problems, this option will likely require a small tree if hard copy is used. This option is best used to learn about the algorithm. A small value for <i>MAXEVL</i> is thus recommended when using <i>msg_lvl</i>=3. Suggested value: 1
56		<p>QJELL's Reference Guide</p> <p>Note: For a given value of <i>msg_lvl</i>, the lower valued options (other than 0) are utilized.</p>

in	<i>seed1</i>	The first seed for the random number generator ranmar_. $0 \leq \text{seed1} \leq 31328$. (integer)
in	<i>seed2</i>	The second seed for the random number generator ranmar_. $0 \leq \text{seed} \leq 30081$. Different values for <i>seed1</i> and <i>seed2</i> will lead to an entirely different sequence of trial points and decisions on downhill moves (when maximizing). See Goffe et al. on how this can be used to test the results of <i>OptimSA</i> .
in	<i>t</i>	On input, the initial temperature. See Goffe et al. for advice. On output, the final temperature.
in	<i>vm</i>	The step length vector. On input it should encompass the region of interest given the starting value <i>x</i> . For point <i>x</i> [<i>i</i>], the next trial point is selected is from <i>x</i> [<i>i</i>]- <i>vm</i> [<i>i</i>] to <i>x</i> [<i>i</i>]+ <i>vm</i> [<i>i</i>]. Since <i>vm</i> is adjusted so that about half of all points are accepted, the input value is not very important (i.e. is the value is off, <i>OptimSA</i> adjusts <i>vm</i> to the correct value).
out	<i>xopt</i>	The variables that optimize the function.
out	<i>fopt</i>	The optimal value of the function.
out	<i>nacc</i>	The number of accepted function evaluations.
out	<i>nfcnev</i>	The total number of function evaluations. In a minor point, note that the first evaluation is not used in the core of the algorithm; it simply initializes the algorithm.
out	<i>nobds</i>	The total number of trial function evaluations that would have been out of bounds of <i>lb</i> and <i>ub</i> . Note that a trial point is randomly selected between <i>lb</i> and <i>ub</i> .

Returns

- 0 - Normal return; termination criteria achieved.
- 1 - Number of function evaluations (*nfcnev*) is greater than the maximum number (*maxevl*).
- 2 - The starting value (*x*) is not inside the bounds (*lb* and *ub*).
- 3 - The initial temperature is not positive.
- 99 - Should not be seen; only used internally.

Template Parameters

<code><OPT_></code>	Class that defined the objective function and its gradient
---------------------------	--

```
int OptimTN ( OPT_ & theOpt, Vect< real_t > & x, Vect< real_t > & low, Vect< real_t >
& up, Vect< int > & pivot, int max_it = -1, real_t toler = 100*OFELI_EPSMCH, int
msg_lvl = 5 )
```

Truncated Newton optimization solver.

Solves a bounds-constrained optimization problem using the Nash's Truncated Newton Algorithm (See paper by S.G. Nash, Newton-type Minimization via the Lanczos method, SIAM J. Numer. Anal. 21 (1984) 770-778). All vector variables are instances of class Vect<real_t>.

Parameters

in	<i>theOpt</i>	Instance of class OPT_ that is implemented by the user and that provides the objective function.
----	---------------	---

in,out	<i>x</i>	Vector that contains an initial guess of the solution and as output the final optimization variables if optimization has succeeded
in	<i>low</i>	Vector of the same size as <i>x</i> that contains for each variable the lower bound to impose. Note that Dirichlet boundary conditions are treated as equality conditions (i.e. lower and upper bounds) and that these ones can be imposed via an auxiliary optimization file (BCAs-Constraint)
in	<i>up</i>	Vector of the same size as <i>x</i> that contains for each variable the upper bound to impose. Note that Dirichlet boundary conditions are treated as equality conditions (i.e. lower and upper bounds) and that these ones can be imposed via an auxiliary optimization function BCAs-Constraint)
in	<i>pivot</i>	Vector of the same size as <i>x</i> that contains on return for each variable an integer value that says if the corresponding constraint was reached (different from 0) or not (= 0). Note that Dirichlet boundary conditions are treated as equality constraints
in	<i>max_it</i>	Maximum number of iterations for convergence
in	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm of projected gradient)
in	<i>msg_lvl</i>	Output message level. Must be between 0 and 10

Returns

Number of performed iterations

Template Parameters

<i><OPT_></i>	Class that provides the objective function. This class is defined by the user.
---------------------	--

The **OPT_** class:

This class is defined by the user. It must have the member function :

```
void Objective(Vect<real_t> &x, real_t &f, Vect<real_t> &g)
```

Here above, *x* is the optimization variable vector, *f* is the value of the objective to calculate for the given *x* and *g* is the gradient vector for *x*.

The function **BCAsConstraint**:

This function is defined by the user:

```
void BCAsConstraint(const Mesh &m, const Vect<real_t> &bc, Vect<real_t> &up, Vect<real_t> &low)
```

This function imposes Dirichlet boundary conditions in an optimization problem as optimization constraints. If such conditions are to be present, this function has to be invoked by giving on input *bc(i)* as the value to impose for the *i*-th optimization variable.

```
int QMR ( const M_ & A, const P1_ & P1, const P2_ & P2, const Vect< real.t > & b, Vect< real.t > & x, int max_it, real.t & toler, int verbose )
```

QMR solver function.

Parameters

in	<i>A</i>	Problem matrix (Instance of abstract class M_).
in	<i>P1</i>	First preconditioner for the Conjugate Gradient Squared.

in	<i>P2</i>	Second preconditioner for the Conjugate Gradient Squared.
in	<i>A</i>	Problem matrix (Instance of abstract class M_).
in	<i>b</i>	Right-hand side vector (class Vect)
in,out	<i>x</i>	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	<i>max_it</i>	Maximum number of iterations.
in	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm).
in	<i>verbose</i>	Information output parameter (0: No output, 1: Output iteration information, 2 and greater : Output iteration information and solution at each iteration.

Returns

Number of iterations:

- 0, Convergence within max.iter iterations
- -1, No convergence after max.iter iterations
- -2, Breakdown in rho
- -3, Breakdown in beta
- -4, Breakdown in gamma
- -5, Breakdown in delta
- -6, Breakdown in ep
- -7, Breakdown in xi

Template Arguments:

- **M_** Matrix storage class
- **P1_** Class of first preconditioner
- **P2_** Class of second preconditioner

int Richardson (const M_ & A, const Vect< T_ > & b, Vect< T_ > & x, real_t omega, int max_it, real_t toler, int verbose)

Richardson solver function.
Parameters

in	<i>A</i>	Problem matrix problem (Instance of abstract class M_).
in	<i>b</i>	Right-hand side vector (class Vect)
	<i>x</i>	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).
in	<i>omega</i>	Relaxation parameter.
in	<i>max_it</i>	Maximum number of iterations.
in,out	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm) in input, effective discrepancy in output.

<code>in</code>	<code>verbose</code>	Information output parameter (0: No output, 1: Output iteration information, 2 and greater: Output iteration information and solution at each iteration.
-----------------	----------------------	--

Returns

`nb_it` Number of performed iterations,

Template Parameters

<code><T_></code>	Data type (real_t, float, complex<real_t>, ...)
<code><M_></code>	Matrix storage class

void Schur (MAT_A_ & A, MAT_U_ & U, MAT_U_ & L, MAT_D_ & D, Vect< T_ > & b, Vect< T_ > & c)

Solve a linear system of equations with a 2x2-block matrix.

The linear system is of the form

$$\begin{bmatrix} A & U \\ L & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

Parameters

<code>in</code>	<code>A</code>	Instance of a matrix class for the first diagonal block. The matrix must be invertible and factorizable (Do not use SpMatrix class) where A, U, L, D are instances of matrix classes,
<code>in</code>	<code>U</code>	Instance of a matrix class for the upper triangle block. The matrix can be rectangular
<code>in</code>	<code>L</code>	Instance of a matrix class for the lower triangle block. The matrix can be rectangular
<code>in</code>	<code>D</code>	Instance of a matrix class for the second diagonal block. The matrix must be factorizable (Do not use SpMatrix class)
<code>in,out</code>	<code>b</code>	Vector (Instance of class Vect) that contains the first block of right-hand side on input and the first block of the solution on output. <code>b</code> must have the same size as the dimension of A.
<code>in,out</code>	<code>c</code>	Vector (Instance of class Vect) that contains the second block of right-hand side on output and the first block of the solution on output. <code>c</code> must have the same size as the dimension of D.

Template Arguments:

Template Parameters

<code><T_></code>	data type (real_t, float, ...)
<code><MAT_A_></code>	Matrix class for A
<code><MAT_U_></code>	Matrix class for D
<code><MAT_D_></code>	Matrix class for U and L

void Schur (Vect< MAT_A_ > & A, Vect< MAT_U_ > & U, Vect< MAT_U_ > & L, MAT_D_ & D, const Vect< Vect< T_ > > & b, const Vect< T_ > & c, Vect< Vect< T_ > > & x, Vect< T_ > & y)

Solve a linear arrow block system by block factorization.

The linear system is of the form

$$\begin{array}{cccc|cccc|c}
 | & A[0] & 0 & \dots & 0 & U[0] & | & | & x[0] & | & | & b[0] & | \\
 | & 0 & A[1] & \dots & 0 & U[1] & | & | & x[1] & | & | & b[1] & | \\
 | & \dots & & & & \dots & | & | & \dots & | & | & \dots & | \\
 | & & & & & & | & | & & | & | & = & | \\
 | & 0 & \dots & 0 & A[nb-1] & U[nb-1] & | & | & x[nb-1] & | & | & b[nb-1] & | \\
 | & L[0] & L[1] & \dots & L[nb-1] & D & | & | & y & | & | & c & |
 \end{array}$$

- Diagonal blocks $A[0], \dots, A[nb-1]$ are instances of template class **MAT_A_**
- Block D is a dense matrix (instance of template class **MAT_D_**)
- Upper rectangular blocks are instances of template class **MAT_U_**
- Lower rectangular blocks are instances of template class **MAT_U_**

All vector blocks are instances of template class **Vect<T_>**

Parameters

in	A	Vect instance whose coefficients are diagonal blocks of the matrix ($A[0], \dots, A[nb-1]$). Each block is an instance of class MAT_A_
in	U	Vect instance whose coefficients are upper last blocks of the matrix ($U[0], \dots, U[nb-1]$). Each block is an instance of class MAT_U_
in	L	Vect instance whose coefficients are lower last blocks of the matrix ($L[0], \dots, L[nb-1]$). Each block is an instance of class MAT_U_
in	D	Instance of class MAT_D_ (A matrix class) that contains last diagonal block of the matrix
in	b	Vect instance whose entries are right-hand side blocks (as instances of class Vect)
in	c	Vect instance that contains the last right-hand side block.
out	x	Vect instance whose entries are solution blocks (as instances of class Vect)
out	y	Vect instance that contains the last solution block.

Template Parameters

$\langle T_-\rangle$	Data type (real_t, float, complex<real_t>, ...)
$\langle MAT_A_ \rangle$	Matrix class for A
$\langle MAT_D_ \rangle$	Matrix class for D
$\langle MAT_U_ \rangle$	Matrix class for U and L

int SSOR (const M_ & A, const Vect< T_ > & b, Vect< T_ > & x, int max_it, double toler, int verbose)

SSOR solver function.

Parameters

in	A	Problem matrix (Instance of abstract class M_).
in	b	Right-hand side vector (class Vect)
in,out	x	Vect instance containing initial solution guess in input and solution of the linear system in output (If iterations have succeeded).

in	<i>max_it</i>	Maximum number of iterations.
in	<i>toler</i>	Tolerance for convergence (measured in relative weighted 2-Norm).
in	<i>verbose</i>	Information output parameter (0: No output, 1: Output iteration information, 2 and greater : Output iteration information and solution at each iteration.

Returns

Number of performed iterations,

Template Arguments:

- *T_* data type (double, float, ...)
- *M_* Matrix storage class

5.16 Utilities

Gathers utility functions and classes.

Files

- file [OFELI.h](#)
Header file that includes all kernel classes of the library.
- file [OFELI_Config.h](#)
File that contains some macros.
- file [constants.h](#)
File that contains some widely used constants.

Classes

- class [Funct](#)
A simple class to parse real valued functions.
- class [Tabulation](#)
To read and manipulate tabulated functions.
- class [UserData< T_ >](#)
Abstract class to define by user various problem data.
- class [Point< T_ >](#)
Defines a point with arbitrary type coordinates.
- class [Point2D< T_ >](#)
Defines a 2-D point with arbitrary type coordinates.
- class [Gauss](#)
Calculate data for Gauss integration.
- class [Timer](#)
To handle elapsed time counting.

Macros

- `#define OFELI_E 2.71828182845904523536028747135`
- `#define OFELI_PI 3.14159265358979323846264338328`
- `#define OFELI_THIRD 0.33333333333333333333333333333333`
- `#define OFELI_SIXTH 0.16666666666666666666666666666667`
- `#define OFELI_TWELVETH 0.08333333333333333333333333333333`
- `#define OFELI_SQRT2 1.41421356237309504880168872421`
- `#define OFELI_SQRT3 1.73205080756887729352744634151`
- `#define OFELI_ONEOVERPI 0.31830988618379067153776752675`
- `#define OFELI_GAUSS2 0.57735026918962576450914878050196`
- `#define OFELI_EPSMCH DBL_EPSILON`
- `#define OFELI_TOLERANCE OFELI_EPSMCH*10000`
- `#define OFELI_IMAG std::complex<double>(0.,1.);`
- `#define PARSE(exp, var) theParser.Parse(exp,var)`
- `#define EVAL(d) theParser.Eval(d)`

Typedefs

- typedef unsigned long `lsize_t`
This type stands for type unsigned long.
- typedef double `real_t`
This type stands for double.
- typedef std::complex< double > `complex_t`
This type stands for type std::complex<double>

Functions

- ostream & `operator<<` (ostream &s, const std::complex< double > &x)
Output a complex number.
- ostream & `operator<<` (ostream &s, const std::string &c)
Output a string.
- template<class T_ >
ostream & `operator<<` (ostream &s, const vector< T_ > &v)
Output a vector instance.
- template<class T_ >
ostream & `operator<<` (ostream &s, const std::pair< T_, T_ > &a)
Output a pair instance.
- void `saveField` (Vect< `real_t` > &v, string output_file, int opt)
Save a vector to an output file in a given file format.
- void `saveField` (Vect< `real_t` > &v, const Grid &g, string output_file, int opt)
Save a vector to an output file in a given file format, for a structured grid data.
- void `saveGnuplot` (string input_file, string output_file, string mesh_file)
Save a vector to an input Gnuplot file.
- void `saveTecplot` (string input_file, string output_file, string mesh_file)
Save a vector to an output file to an input Tecplot file.
- void `saveVTK` (string input_file, string output_file, string mesh_file)
Save a vector to an output VTK file.
- void `saveGmsh` (string input_file, string output_file, string mesh_file)
Save a vector to an output Gmsh file.
- ostream & `operator<<` (ostream &s, const Tabulation &t)
Output Tabulated function data.
- template<class T_ >
bool `operator==` (const Point< T_ > &a, const Point< T_ > &b)
Operator ==
- template<class T_ >
Point< T_ > `operator+` (const Point< T_ > &a, const Point< T_ > &b)
Operator +
- template<class T_ >
Point< T_ > `operator+` (const Point< T_ > &a, const T_ &x)
Operator +
- template<class T_ >
Point< T_ > `operator-` (const Point< T_ > &a)
Unary Operator -
- template<class T_ >
Point< T_ > `operator-` (const Point< T_ > &a, const Point< T_ > &b)

- Operator -*
 - `template<class T_ >`
`Point< T_ > operator-` (`const Point< T_ > &a`, `const T_ &x`)
 - Operator -*
 - `template<class T_ >`
`Point< T_ > operator*` (`const T_ &a`, `const Point< T_ > &b`)
 - Operator **
 - `template<class T_ >`
`Point< T_ > operator*` (`const int &a`, `const Point< T_ > &b`)
 - Operator *.*
 - `template<class T_ >`
`Point< T_ > operator*` (`const Point< T_ > &b`, `const T_ &a`)
 - Operator /*
 - `template<class T_ >`
`Point< T_ > operator*` (`const Point< T_ > &b`, `const int &a`)
 - Operator **
 - `template<class T_ >`
`T_ operator*` (`const Point< T_ > &a`, `const Point< T_ > &b`)
 - Operator **
 - `template<class T_ >`
`Point< T_ > operator/` (`const Point< T_ > &b`, `const T_ &a`)
 - Operator /*
 - `bool areClose` (`const Point< double > &a`, `const Point< double > &b`, `double toler=OFELI.T-
OLERANCE`)
 - Return true if both instances of class Point<double> are distant with less then toler*
 - `double SqrDistance` (`const Point< double > &a`, `const Point< double > &b`)
 - Return squared euclidean distance between points a and b*
 - `double Distance` (`const Point< double > &a`, `const Point< double > &b`)
 - Return euclidean distance between points a and b*
 - `template<class T_ >`
`std::ostream & operator<<` (`std::ostream &s`, `const Point< T_ > &a`)
 - Output point coordinates.*
 - `template<class T_ >`
`bool operator==` (`const Point2D< T_ > &a`, `const Point2D< T_ > &b`)
 - Operator ==.*
 - `template<class T_ >`
`Point2D< T_ > operator+` (`const Point2D< T_ > &a`, `const Point2D< T_ > &b`)
 - Operator +.*
 - `template<class T_ >`
`Point2D< T_ > operator+` (`const Point2D< T_ > &a`, `const T_ &x`)
 - Operator +.*
 - `template<class T_ >`
`Point2D< T_ > operator-` (`const Point2D< T_ > &a`)
 - Unary Operator -*
 - `template<class T_ >`
`Point2D< T_ > operator-` (`const Point2D< T_ > &a`, `const Point2D< T_ > &b`)
 - Operator -*
 - `template<class T_ >`
`Point2D< T_ > operator-` (`const Point2D< T_ > &a`, `const T_ &x`)

Operator -

- `template<class T_ >`
`Point2D< T_ > operator*` (const T_ &a, const Point2D< T_ > &b)

*Operator *.*

- `template<class T_ >`
`Point2D< T_ > operator*` (const int &a, const Point2D< T_ > &b)
- `template<class T_ >`
`Point2D< T_ > operator*` (const Point2D< T_ > &b, const T_ &a)

Operator /

- `template<class T_ >`
`Point2D< T_ > operator*` (const Point2D< T_ > &b, const int &a)

*Operator **

- `template<class T_ >`
`T_ operator*` (const Point2D< T_ > &b, const Point2D< T_ > &a)

*Operator *.*

- `template<class T_ >`
`Point2D< T_ > operator/` (const Point2D< T_ > &b, const T_ &a)

Operator /

- `bool areClose` (const Point2D< `real_t` > &a, const Point2D< `real_t` > &b, `real_t` toler=`OFELI_TOLERANCE`)

Return true if both instances of class `Point2D<real_t>` are distant with less then toler [Default: `OFELI_EPSMCH`].

- `real_t SqrDistance` (const Point2D< `real_t` > &a, const Point2D< `real_t` > &b)

Return squared euclidean distance between points a and b

- `real_t Distance` (const Point2D< `real_t` > &a, const Point2D< `real_t` > &b)

Return euclidean distance between points a and b

- `template<class T_ >`
`std::ostream & operator<<` (std::ostream &s, const Point2D< T_ > &a)

Output point coordinates.

- `void getMesh` (string file, ExternalFileFormat form, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in an external file format.

- `void getBamg` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Bamg` format.

- `void getEasymesh` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Easymesh` format.

- `void getGambit` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Gambit` neutral format.

- `void getGmsh` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Gmsh` format.

- `void getMatlab` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a Matlab mesh data.

- `void getNetgen` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Netgen` format.

- `void getTetgen` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Tetgen` format.

- `void getTriangle` (string file, Mesh &mesh, size_t nb_dof=1)

Construct an instance of class `Mesh` from a mesh file stored in `Triangle` format.

- `void saveMesh` (const string &file, const Mesh &mesh, ExternalFileFormat form)

- This function saves mesh data a file for a given external format.*

 - void `saveGmsh` (const string &gp_file, const Mesh &mesh)

This function outputs a `Mesh` instance in a file in `Gmsh` format.

 - void `saveGnuplot` (const string &file, const Mesh &mesh)

This function outputs a `Mesh` instance in a file in `Gmsh` format.

 - void `saveMatlab` (const string &file, const Mesh &mesh)

This function outputs a `Mesh` instance in a file in `Matlab` format.

 - void `saveTecplot` (const string &file, const Mesh &mesh)

This function outputs a `Mesh` instance in a file in `Tecplot` format.

 - void `saveVTK` (const string &file, const Mesh &mesh)

This function outputs a `Mesh` instance in a file in `VTK` format.

 - void `saveBamg` (const string &file, Mesh &mesh)

This function outputs a `Mesh` instance in a file in `Bamg` format.

 - void `BSpline` (size_t n, size_t t, Vect< Point< `real_t` > > &control, Vect< Point< `real_t` > > &output, size_t num_output)

Function to perform a B-spline interpolation.

 - void `banner` (const string &prog=" ")

Outputs a banner as header of any developed program.

 - template<class T_ >
void `QuickSort` (std::vector< T_ > &a, int begin, int end)

Function to sort a vector.

 - template<class T_ >
void `qksort` (std::vector< T_ > &a, int begin, int end)

Function to sort a vector.

 - template<class T_ , class C_ >
void `qksort` (std::vector< T_ > &a, int begin, int end, C_ compare)

Function to sort a vector according to a key function.

 - int `Sgn` (`real_t` a)

Return sign of a: -1 or 1.

 - `real_t` `Abs2` (`complex_t` a)

Return square of modulus of complex number a

 - `real_t` `Abs2` (`real_t` a)

Return square of real number a

 - `real_t` `Abs` (`real_t` a)

Return absolute value of a

 - `real_t` `Abs` (`complex_t` a)

Return modulus of complex number a

 - `real_t` `Abs` (const Point< `real_t` > &p)

Return Norm of vector a

 - `real_t` `Conjg` (`real_t` a)

Return complex conjugate of real number a

 - `complex_t` `Conjg` (`complex_t` a)

Return complex conjugate of complex number a

 - `real_t` `Max` (`real_t` a, `real_t` b, `real_t` c)

Return maximum value of real numbers a, b and c

 - int `Max` (int a, int b, int c)

Return maximum value of integer numbers a, b and c

- `real_t Min (real_t a, real_t b, real_t c)`
Return minimum value of real numbers a , b and c
- `int Min (int a, int b, int c)`
Return minimum value of integer numbers a , b and c
- `real_t Max (real_t a, real_t b, real_t c, real_t d)`
Return maximum value of integer numbers a , b , c and d
- `int Max (int a, int b, int c, int d)`
Return maximum value of integer numbers a , b , c and d
- `real_t Min (real_t a, real_t b, real_t c, real_t d)`
Return minimum value of real numbers a , b , c and d
- `int Min (int a, int b, int c, int d)`
Return minimum value of integer numbers a , b , c and d
- `real_t Arg (complex_t x)`
Return argument of complex number x
- `complex_t Log (complex_t x)`
Return principal determination of logarithm of complex number x
- `template<class T_ >`
`T_ Sqr (T_ x)`
Return square of value x
- `template<class T_ >`
`void Scale (T_ a, const vector< T_ > &x, vector< T_ > &y)`
Multiply vector x by a and save result in vector y
- `template<class T_ >`
`void Scale (T_ a, vector< T_ > &x)`
Multiply vector x by a
- `template<class T_ >`
`void Xpy (size_t n, T_ *x, T_ *y)`
Add array x to y
- `template<class T_ >`
`void Xpy (const vector< T_ > &x, vector< T_ > &y)`
Add vector x to y
- `template<class T_ >`
`void Axy (size_t n, T_ a, T_ *x, T_ *y)`
Multiply array x by a and add result to y
- `template<class T_ >`
`void Axy (T_ a, const vector< T_ > &x, vector< T_ > &y)`
Multiply vector x by a and add result to y
- `template<class T_ >`
`void Copy (size_t n, T_ *x, T_ *y)`
Copy array x to y n is the arrays size.
- `real_t Error2 (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute L^2 error between vectors x and y
- `real_t RError2 (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute L^2 error between vectors x and y
- `real_t ErrorMax (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute Max. error between vectors x and y
- `real_t RErrorMax (const vector< real_t > &x, const vector< real_t > &y)`

- Return relative Max. error between vectors x and y*

 - `template<class T_ >`
`T_ Dot (size_t n, T_ *x, T_ *y)`
Return dot product of arrays x and y
 - `template<class T_ >`
`T_ Dot (const vector< T_ > &x, const vector< T_ > &y)`
Return dot product of vectors x and y .
 - `real_t Dot (const vector< real_t > &x, const vector< real_t > &y)`
Return dot product of vectors x and y
 - `template<class T_ >`
`T_ Dot (const Point< T_ > &x, const Point< T_ > &y)`
Return dot product of x and y
 - `template<class T_ >`
`void Clear (vector< T_ > &v)`
Assign 0 to all entries of a vector.
 - `real_t Nrm2 (size_t n, real_t *x)`
Return 2-norm of array x
 - `real_t Nrm2 (const vector< real_t > &x)`
Return 2-norm of vector x
 - `template<class T_ >`
`real_t Nrm2 (const Point< T_ > &a)`
Return 2-norm of a
 - `bool Equal (real_t x, real_t y, real_t toler=OFELI_EPSMCH)`
Function to return true if numbers x and y are close up to a given tolerance $toler$
 - `char itoc (int i)`
Function to convert an integer to a character.
 - `string itos (int i)`
Function to convert an integer to a string.
 - `string dtos (real_t d)`
Function to convert a real to a string.
 - `template<class T_ >`
`T_ stringTo (const std::string &s)`
Function to convert a string to a template type parameter.
 - `void RTrim (char *s)`
Function to remove blanks at the end of a string.
 - `void LTrim (char *s)`
Function to remove blanks at the beginning of a string.
 - `void Trim (char *s)`
Function to remove blanks at the beginning and end of a string.
 - `template<class T_ >`
`void Swap (T_ &a, T_ &b)`
Swap elements a and b .

5.16.1 Detailed Description

Gathers utility functions and classes.

5.16.2 Macro Definition Documentation

#define OFELI_E 2.71828182845904523536028747135

Value of e or \exp (with 28 digits)

#define OFELI_PI 3.14159265358979323846264338328

Value of Pi (with 28 digits)

#define OFELI_THIRD 0.333333333333333333333333333333

Value of $1/3$ (with 28 digits)

#define OFELI_SIXTH 0.166666666666666666666666666667

Value of $1/6$ (with 28 digits)

#define OFELI_TWELVETH 0.083333333333333333333333333333

Value of $1/12$ (with 28 digits)

#define OFELI_SQRT2 1.41421356237309504880168872421

Value of $\sqrt{2}$ (with 28 digits)

#define OFELI_SQRT3 1.73205080756887729352744634151

Value of $\sqrt{3}$ (with 28 digits)

#define OFELI_ONEOVERPI 0.31830988618379067153776752675

Value of $1/Pi$ (with 28 digits)

#define OFELI_GAUSS2 0.57735026918962576450914878050196

Value of $1/\sqrt{3}$ (with 32 digits)

#define OFELI_EPSMCH DBL_EPSILON

Value of Machine Epsilon

#define OFELI_TOLERANCE OFELI_EPSMCH*10000

Default tolerance for an iterative process = $OFELI_EPSMCH * 10000$

#define OFELI_IMAG std::complex<double>(0.,1.);

= Unit imaginary number (i)

#define PARSE(exp, var) theParser.Parse(exp,var)

A macro that parses a regular expression exp using the variables in the string var . For instance, to parse the function $\sin(x+y)$ one must declare $PARSE("sin(x+y)","x,y")$

#define EVAL(d) theParser.Eval(d)

A macro that evaluates a parsed regular expression For instance, with a declaration `PARSE("sin(x+y)","x,y")` the data `x=1` and `y=2` using this function must be evaluated as follows: `EVAL(d)` with `d[0]=1`, `d[1]=2`

5.16.3 Function Documentation**void saveField (Vect< real_t > & v, string output_file, int opt)**

Save a vector to an output file in a given file format.

Case where the vector contains mesh information

Parameters

in	v	Vect instance to save
in	output_file	Output file where to save the vector
in	opt	Option to choose file format to save. This is to be chosen among enumerated values: GMSH GNUPLOT MATLAB TECPLOT VTK

void saveField (Vect< real_t > & v, const Grid & g, string output_file, int opt = VTK)

Save a vector to an output file in a given file format, for a structured grid data.

Parameters

in	v	Vect instance to save
in	g	Grid instance
in	output_file	Output file where to save the vector
in	opt	Option to choose file format to save. This is to be chosen among enumerated values: VTK

void saveGnuplot (string input_file, string output_file, string mesh_file)

Save a vector to an input Gnuplot file.

Gnuplot is a command-line driven program for producing 2D and 3D plots. It is under the GNU General Public License. Available information can be found in the site:

<http://www.gnuplot.info/>

Parameters

in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)
in	mesh_file	File containing mesh data

void saveTecplot (string input_file, string output_file, string mesh_file)

Save a vector to an output file to an input Tecplot file.

Tecplot is high quality post graphical commercial processing program developed by Amtec. Available information can be found in the site: <http://www.tecplot.com>

Parameters

in	input_file	Input file (OFELI XML file containing a field).
in	output_file	Output file (gnuplot format file)

<i>in</i>	<i>mesh_file</i>	File containing mesh data
-----------	------------------	---------------------------

saveVTK (string input_file, string output_file, string mesh_file)

Save a vector to an output **VTK** file.

The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics. Available information can be found in the site:

<http://public.kitware.com/VTK/>

Parameters

<i>in</i>	<i>input_file</i>	Input file (OFELI XML file containing a field).
<i>in</i>	<i>output_file</i>	Output file (VTK format file)
<i>in</i>	<i>mesh_file</i>	File containing mesh data

void saveGmsh (string input_file, string output_file, string mesh_file)

Save a vector to an output **Gmsh** file.

Gmsh is a free mesh generator and postprocessor that can be downloaded from the site:

<http://www.geuz.org/gmsh/>

Parameters

<i>in</i>	<i>input_file</i>	Input file (OFELI XML file containing a field).
<i>in</i>	<i>output_file</i>	Output file (Gmsh format file)
<i>in</i>	<i>mesh_file</i>	File containing mesh data

bool operator== (const Point< T_ > & a, const Point< T_ > & b)

Operator ==

Return true if a=b, false if not.

Point< T_ > operator+ (const Point< T_ > & a, const Point< T_ > & b)

Operator +

Return sum of two points a and b

Point< T_ > operator+ (const Point< T_ > & a, const T_ & x)

Operator +

Translate a by x

Point< T_ > operator- (const Point< T_ > & a)

Unary Operator -

Return minus a

Point< T_ > operator- (const Point< T_ > & a, const Point< T_ > & b)

Operator -

Return point a minus point b

Point< T_ > operator- (const Point< T_ > & a, const T_ & x)

Operator -

Translate a by -x

Point< T_ > operator* (const T_ & a, const Point< T_ > & b)

Operator *

Return point b premultiplied by constant a

Point< T_ > operator* (const int & a, const Point< T_ > & b)

Operator *.

Return point b divided by integer constant a

Point< T_ > operator* (const Point< T_ > & b, const T_ & a)

Operator /

Return point b multiplied by constant a

Point< T_ > operator* (const Point< T_ > & b, const int & a)

Operator *

Return point b postmultiplied by constant a

T_ operator* (const Point< T_ > & b, const Point< T_ > & a)

Operator *

Return inner (scalar) product of points a and b

Point< T_ > operator/ (const Point< T_ > & b, const T_ & a)

Operator /

Return point b divided by constant a

bool operator== (const Point2D< T_ > & a, const Point2D< T_ > & b)

Operator ==.

Return true if a=b, false if not.

Point2D< T_ > operator+ (const Point2D< T_ > & a, const Point2D< T_ > & b)

Operator +.

Return sum of two points a and b

Point2D< T_ > operator+ (const Point2D< T_ > & a, const T_ & x)

Operator +.

Translate a by x

Point2D< T_ > operator- (const Point2D< T_ > & a)

Unary Operator -

Return minus a

Point2D< T_ > operator- (const Point2D< T_ > & a, const Point2D< T_ > & b)

Operator -

Return point a minus point b

Point2D< T_ > operator- (const Point2D< T_ > & a, const T_ & x)

Operator -

Translate a by $-x$

Point2D< T_ > operator* (const T_ & a, const Point2D< T_ > & b)

Operator *.

Return point b premultiplied by constant a

Point2D< T_ > operator* (const int & a, const Point2D< T_ > & b)

Operator *.

Return point b divided by integer constant a

Point2D< T_ > operator* (const Point2D< T_ > & b, const T_ & a)

Operator /

Return point b postmultiplied by constant a

Point2D< T_ > operator* (const Point2D< T_ > & b, const int & a)

Operator *

Return point b postmultiplied by constant a

T_ operator* (const Point2D< T_ > & b, const Point2D< T_ > & a)

Operator *.

Return point b postmultiplied by integer constant a .

Point2D< T_ > operator/ (const Point2D< T_ > & b, const T_ & a)

Operator /

Return point b divided by constant a

void getMesh (string file, ExternalFileFormat form, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class [Mesh](#) from a mesh file stored in an external file format.

Parameters

in	file	Input mesh file name.
in	form	<p>Format of the mesh file. This one can be chosen among the enumerated values:</p> <ul style="list-style-type: none"> • GMSH: Mesh generator Gmsh, see site: http://www.geuz.org/gmsh/ • MATLAB: Matlab file, see site: http://www.mathworks.com/products/matlab/ • EASYMESH: Easymesh is a 2-D mesh generator, see site: http://web.mit.edu/easymesh_v1.4/www/easymesh.html • GAMBIT: Gambit is a mesh generator associated to Fluent http://www.stanford.edu/class/me469b/gambit-download.html • BAMG: Mesh generator Bamg, see site: http://raweb.inria.fr/rapportsactivite/RA2002/gamma/uid25.html • NETGEN: Netgen is a 3-D mesh generator, see site: http://www.hpfem.jku.at/netgen/ • TETGEN: Tetgen is a 3-D mesh generator, see site: http://tetgen.berlios.de/ • TRIANGLE_FF: Triangle is a 2-D mesh generator, see site: http://www.cs.cmu.edu/~quake/triangle.html
out	mesh	Mesh instance created by the function.
in	nb_dof	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

void getBamg (string file, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class [Mesh](#) from a mesh file stored in [Bamg](#) format.

Parameters

in	file	Name of a file written in the Bamg format.
----	------	--

Note

Bamg is a 2-D mesh generator. It allows to construct adapted meshes from a given metric. It was developed at INRIA, France. Available information can be found in the site:

<http://raweb.inria.fr/rapportsactivite/RA2002/gamma/uid25.html>

Parameters

out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

```
void getEasymesh ( string file, Mesh & mesh, size_t nb_dof = 1 )
```

Construct an instance of class [Mesh](#) from a mesh file stored in [Easymesh](#) format.

Parameters

in	<i>file</i>	Name of a file (without extension) written in Easymesh format. Actually, the function <code>Easymesh2MDF</code> attempts to read mesh data from files <code>file.e</code> , <code>file.n</code> and <code>file.s</code> produced by Easymesh .
----	-------------	--

Note

Easymesh is a free program that generates 2-D, unstructured, Delaunay and constrained Delaunay triangulations in general domains. It can be downloaded from the site:

<http://www.dinma.univ.trieste.it/nirftc/research/easymesh/Default.htm>

Parameters

in	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

```
void getGambit ( string file, Mesh & mesh, size_t nb_dof = 1 )
```

Construct an instance of class [Mesh](#) from a mesh file stored in **Gambit** neutral format.

Note

Gambit is a commercial mesh generator associated to the CFD code [Fluent](#). Informations about **Gambit** can be found in the site:

<http://www.fluent.com/software/gambit/>

Parameters

in	<i>file</i>	Name of a file written in the Gambit neutral format.
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

```
void getGmsh ( string file, Mesh & mesh, size_t nb_dof = 1 )
```

Construct an instance of class [Mesh](#) from a mesh file stored in [Gmsh](#) format.

Note

Gmsh is a free mesh generator that can be downloaded from the site:

<http://www.geuz.org/gmsh/>

Parameters

in	<i>file</i>	Name of a file written in the Gmsh format.
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

void getMatlab (string file, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class **Mesh** from a Matlab mesh data.

Note

Matlab is a language of scientific computing including visualization. It is developed by **Math-Works**. Available information can be found in the site:
<http://www.mathworks.com/products/matlab/>

Parameters

in	<i>file</i>	Name of a file created by Matlab by executing the script file <code>Matlab2-OFELI.m</code>
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

void getNetgen (string file, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class **Mesh** from a mesh file stored in **Netgen** format.

Note

Netgen is a tetrahedral mesh generator that can be downloaded from the site:
<http://www.hpfem.jku.at/netgen/>

Parameters

in	<i>file</i>	Name of a file written in the Netgen format.
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. [default = 1]

void getTetgen (string file, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class **Mesh** from a mesh file stored in **Tetgen** format.

Note

Tetgen is a free three-dimensional mesh generator that can be downloaded in the site:
<http://tetgen.berlios.de/>

Parameters

in	<i>file</i>	Name of a file written in the Tetgen format.
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

void getTriangle (string file, Mesh & mesh, size_t nb_dof = 1)

Construct an instance of class [Mesh](#) from a mesh file stored in **Triangle** format.

Note

TRIANGLE is a C program that can generate meshes, Delaunay triangulations and Voronoi diagrams for 2D pointsets that can be downloaded in the site:

http://people.scs.fsu.edu/~burkardt/c_src/triangle/triangle.html/

Parameters

in	<i>file</i>	Name of a file written in the Tetgen format.
out	<i>mesh</i>	Mesh instance created by the function.
in	<i>nb_dof</i>	Number of degrees of freedom for each node. This information is not provided, in general, by mesh generators. Its default value here is 1.

void saveMesh (const string & file, const Mesh & mesh, ExternalFileFormat form)

This function saves mesh data a file for a given external format.

Parameters

in	<i>file</i>	File where to store mesh
in	<i>mesh</i>	Mesh instance to save
in	<i>form</i>	Format of the mesh file. This one can be chosen among the enumerated values: <ul style="list-style-type: none"> • GMSH: Mesh generator and graphical postprocessor Gmsh: http://www.geuz.org/gmsh/ • GNUPLOT: Well known graphics software: http://www.gnuplot.info/ • MATLAB: Matlab file: http://www.mathworks.com/products/matlab/ • TECPLOT: Commercial graphics software: http://www.tecplot.com • VTK: Graphics format for the free postprocessor ParaView: http://public.kitware.com/VTK/

void saveGmsh (const string & file, const Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **Gmsh** format.

Note

Gmsh is a free mesh generator that can be downloaded from the site: <http://www.geuz.org/gmsh/>

Parameters

out	<i>file</i>	Output file in Gmsh format.
in	<i>mesh</i>	Mesh instance to save.

void saveGnuplot (const string & file, const Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **Gmsh** format.

Note

Gnuplot is a command-line driven program for producing 2D and 3D plots. It is under the GNU General Public License. Available information can be found in the site:

<http://www.gnuplot.info/>

Parameters

out	<i>file</i>	Output file in Gnuplot format.
in	<i>mesh</i>	Mesh instance to save.

void saveMatlab (const string & file, const Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **Matlab** format.

Note

Matlab is a language of scientific computing including visualization. It is developed by **Math-Works**. Available information can be found in the site:

<http://www.mathworks.com/products/matlab/>

Parameters

out	<i>file</i>	Output file in Matlab format.
in	<i>mesh</i>	Mesh instance to save.

void saveTecplot (const string & file, const Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **Tecplot** format.

Note

Tecplot is high quality post graphical commercial processing program developed by **Amtec**. Available information can be found in the site:

<http://www.tecplot.com>

Parameters

out	<i>file</i>	Output file in Tecplot format.
in	<i>mesh</i>	Mesh instance to save.

void saveVTK (const string & file, const Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **VTK** format.

Note

The Visualization ToolKit (VTK) is an open source, freely available software system for 3D computer graphics. Available information can be found in the site:

<http://public.kitware.com/VTK/>

Parameters

out	file	Output file in VTK format.
in	mesh	Mesh instance to save.

void saveBamg (const string & file, Mesh & mesh)

This function outputs a [Mesh](#) instance in a file in **Bamg** format.

Parameters

in	file	Name of a file written in the Bamg format.
----	------	--

Note

Bamg is a 2-D mesh generator. It allows to construct adapted meshes from a given metric. It was developed at INRIA, France. Available information can be found in the site:

<http://raweb.inria.fr/rapportsactivite/RA2002/gamma/uid25.html>

Parameters

in	mesh	Mesh instance.
----	------	--------------------------------

BSpline (size_t n, size_t t, Vect< Point< real_t > > & control, Vect< Point< real_t > > & output, size_t num_output)

Function to perform a B-spline interpolation.

This program is adapted from a free program distributed by Keith Vertanen (vertankd@cda.mrs.-umn.edu) in 1994.

Parameters

in	n	Number of control points minus 1.
in	t	Degree of the polynomial plus 1.
in	control	Control point array made up of Point structure.
out	output	Vector in which the calculated spline points are to be put.
in	num_output	How many points on the spline are to be calculated.

Note

Condition: $n+2 > t$ (No curve results if $n+2 \leq t$) Control vector contains the number of points specified by n Output array is the proper size to hold num_output point structures

void banner (const string & prog = " ")

Outputs a banner as header of any developed program.

Parameters

in	prog	Calling program name. Enables writing a copyright notice accompanying the program.
----	------	--

void QuickSort (std::vector< T_ > & a, int begin, int end)

Function to sort a vector.

qksort uses the famous quick sorting algorithm.

Parameters

in,out	<i>a</i>	Vector to sort.
in	<i>begin</i>	index of starting iterator
in	<i>end</i>	index of ending iterator

The calling program must provide an overloading of the operator < for the type **T₋**.

void qksort (std::vector< T₋ > & a, int begin, int end)

Function to sort a vector.

qksort uses the famous quick sorting algorithm.

Parameters

in,out	<i>a</i>	Vector to sort.
in	<i>begin</i>	index of starting index (default value is 0)
in	<i>end</i>	index of ending index (default value is the vector size - 1)

void qksort (std::vector< T₋ > & a, int begin, int end, C₋ compare)

Function to sort a vector according to a key function.

qksort uses the famous quick sorting algorithm.

Parameters

in,out	<i>a</i>	Vector to sort.
in	<i>begin</i>	index of starting index (0 for the beginning of the vector)
in	<i>end</i>	index of ending index
in	<i>compare</i>	A function object that implements the ordering. The user must provide this function that returns a boolean function that is true if the first argument is less than the second and false if not.

void Scale (T₋ a, const vector< T₋ > & x, vector< T₋ > & y)

Multiply vector x by a and save result in vector y

x and y are instances of class vector<T₋>

void Scale (T₋ a, vector< T₋ > & x)

Multiply vector x by a

x is an instance of class vector<T₋>

void Xpy (const vector< T₋ > & x, vector< T₋ > & y)

Add vector x to y

x and y are instances of class vector<T₋>

void Axy (size_t n, T₋ a, T₋ * x, T₋ * y)

Multiply array x by a and add result to y

n is the arrays size.

void Axy (T₋ a, const vector< T₋ > & x, vector< T₋ > & y)

Multiply vector x by a and add result to y

x and y are instances of class vector<T₋>

T_ Dot (size_t n, T_ * x, T_ * y)

Return dot product of arrays x and y
n is the arrays size.

T_ Dot (const vector< T_ > & x, const vector< T_ > & y)

Return dot product of vectors x and y.
x and y are instances of class vector<T_>

real_t Dot (const vector< real_t > & x, const vector< real_t > & y)

Return dot product of vectors x and y
x and y are instances of class vector<T_>

void Clear (vector< T_ > & v)

Assign 0 to all entries of a vector.
Parameters

in	v	Vector to clear
----	---	-----------------

real_t Nrm2 (size_t n, real_t * x)

Return 2-norm of array x
Parameters

in	n	is Array length
in	x	Array to treat

bool Equal (real_t x, real_t y, real_t toler = OFELI_EPSMCH)

Function to return true if numbers x and y are close up to a given tolerance toler
Default value of tolerance is the constant OFELI_EPSMCH

5.17 Vector and Matrix

Gathers vector and matrix related classes.

Classes

- class `BMatrix< T_ >`
To handle band matrices.
- class `DMatrix< T_ >`
To handle dense matrices.
- class `DSMatrix< T_ >`
To handle symmetric dense matrices.
- class `LocalMatrix< T_, NR_, NC_ >`
Handles small size matrices like element matrices, with a priori known size.
- class `LocalVect< T_, N_ >`
Handles small size vectors like element vectors.
- class `SkMatrix< T_ >`
To handle square matrices in skyline storage format.
- class `SkSMatrix< T_ >`
To handle symmetric matrices in skyline storage format.
- class `SpMatrix< T_ >`
To handle matrices in sparse storage format.
- class `TrMatrix< T_ >`
To handle tridiagonal matrices.
- class `Vect< T_ >`
To handle general purpose vectors.

Functions

- `template<class T_ >`
`BMatrix< T_ > operator* (T_ a, const BMatrix< T_ > &A)`
*Operator * (Premultiplication of matrix by constant)*
- `template<class T_ >`
`ostream & operator<< (ostream &s, const BMatrix< T_ > &a)`
Output matrix in output stream.
- `template<class T_ >`
`ostream & operator<< (ostream &s, const DMatrix< T_ > &a)`
Output matrix in output stream.
- `template<class T_ >`
`ostream & operator<< (ostream &s, const DSMatrix< T_ > &a)`
Output matrix in output stream.
- `template<class T_ , size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator* (T_ a, const LocalMatrix< T_, NR_, NC_ > &x)`
*Operator * (Multiply matrix x by scalar a)*
- `template<class T_ , size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator/ (T_ a, const LocalMatrix< T_, NR_, NC_ > &x)`
Operator / (Divide matrix x by scalar a)

- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator+ (const LocalMatrix< T_, NR_, NC_ > &x, const LocalMatrix< T_, NR_, NC_ > &y)`
Operator + (Add matrix x to y)
- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator- (const LocalMatrix< T_, NR_, NC_ > &x, const LocalMatrix< T_, NR_, NC_ > &y)`
Operator - (Subtract matrix y from x)
- `template<class T_, size_t NR_, size_t NC_>`
`ostream & operator<< (ostream &s, const LocalMatrix< T_, NR_, NC_ > &a)`
Output vector in output stream.
- `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator+ (const LocalVect< T_, N_ > &x, const LocalVect< T_, N_ > &y)`
Operator + (Add two vectors)
- `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator- (const LocalVect< T_, N_ > &x, const LocalVect< T_, N_ > &y)`
Operator - (Subtract two vectors)
- `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator* (T_ a, const LocalVect< T_, N_ > &x)`
*Operator * (Premultiplication of vector by constant)*
- `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator/ (T_ a, const LocalVect< T_, N_ > &x)`
Operator / (Division of vector by constant)
- `template<class T_, size_t N_>`
`double Dot (const LocalVect< T_, N_ > &a, const LocalVect< T_, N_ > &b)`
Calculate dot product of 2 vectors (instances of class [LocalVect](#))
- `template<class T_, size_t N_>`
`void Scale (T_ a, const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
Multiply vector x by constant a and store result in y.
- `template<class T_, size_t N_>`
`void Scale (T_ a, LocalVect< T_, N_ > &x)`
Multiply vector x by constant a and store result in x.
- `template<class T_, size_t N_>`
`void Axp (T_ a, const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
*Add a*x to vector y.*
- `template<class T_, size_t N_>`
`void Copy (const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
Copy vector x into vector y.
- `template<class T_, size_t N_>`
`ostream & operator<< (ostream &s, const LocalVect< T_, N_ > &v)`
Output vector in output stream.
- `template<class T_>`
`ostream & operator<< (ostream &s, const SkMatrix< T_ > &a)`
Output matrix in output stream.
- `template<class T_>`
`ostream & operator<< (ostream &s, const SkSMatrix< T_ > &a)`
Output matrix in output stream.

- `template<class T_>`
`TrMatrix< T_ > operator* (T_ a, const TrMatrix< T_ > &A)`
*Operator * (Premultiplication of matrix by constant)*
- `template<class T_>`
`ostream & operator<< (ostream &s, const TrMatrix< T_ > &a)`
Output matrix in output stream.

5.17.1 Detailed Description

Gathers vector and matrix related classes.

5.17.2 Function Documentation

BMatrix< T_ > operator* (T_ a, const BMatrix< T_ > & A)

Operator * (Premultiplication of matrix by constant)

Returns

$a * A$

LocalMatrix< T_, NR_, NC_ > operator* (T_ a, const LocalMatrix< T_, NR_, NC_ > & x)

Operator * (Multiply matrix x by scalar a)

Returns

$a * x$

LocalMatrix< T_, NR_, NC_ > operator/ (T_ a, const LocalMatrix< T_, NR_, NC_ > & x)

Operator / (Divide matrix x by scalar a)

Returns

x / a

LocalMatrix< T_, NR_, NC_ > operator+ (const LocalMatrix< T_, NR_, NC_ > & x, const LocalMatrix< T_, NR_, NC_ > & y)

Operator + (Add matrix x to y)

Returns

$x + y$

LocalMatrix< T_, NR_, NC_ > operator- (const LocalMatrix< T_, NR_, NC_ > & x, const LocalMatrix< T_, NR_, NC_ > & y)

Operator - (Subtract matrix y from x)

Returns

$x - y$

LocalVect< T_, N_ > operator+ (const LocalVect< T_, N_ > & x, const LocalVect< T_, N_ > & y)

Operator + (Add two vectors)

Returns

$x+y$

LocalVect< T_, N_ > operator- (const LocalVect< T_, N_ > & x, const LocalVect< T_, N_ > & y)

Operator - (Subtract two vectors)

Returns

$x-y$

LocalVect< T_, N_ > operator* (T_ a, const LocalVect< T_, N_ > & x)

Operator * (Premultiplication of vector by constant)

Returns

$a*x$

LocalVect< T_, N_ > operator/ (T_ a, const LocalVect< T_, N_ > & x)

Operator / (Division of vector by constant)

Returns

x/a

double Dot (const LocalVect< T_, N_ > & a, const LocalVect< T_, N_ > & b)

Calculate dot product of 2 vectors (instances of class [LocalVect](#))

Returns

Dot product

TrMatrix< T_ > operator* (T_ a, const TrMatrix< T_ > & A)

Operator * (Premultiplication of matrix by constant)

Returns

$a*A$

Chapter 6

Namespace Documentation

6.1 OFELI Namespace Reference

A namespace to group all library classes, functions, ...

Classes

- class [SkMatrix](#)
To handle square matrices in skyline storage format.
- class [SkSMatrix](#)
To handle symmetric matrices in skyline storage format.
- class [SpMatrix](#)
To handle matrices in sparse storage format.
- class [AbsEqua](#)
Mother abstract class to describe equation.
- class [LocalVect](#)
Handles small size vectors like element vectors.
- class [ICPG1D](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.
- class [ICPG2DT](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.
- class [ICPG3DT](#)
Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D.
- class [LCL1D](#)
Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.
- class [LCL2DT](#)
Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.
- class [LCL3DT](#)
Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.
- class [Muscl](#)
Parent class for hyperbolic solvers with Muscl scheme.
- class [Vect](#)
To handle general purpose vectors.
- class [Muscl1D](#)

- Class for 1-D hyperbolic solvers with [Muscl](#) scheme.*

 - class [Muscl2DT](#)
- Class for 2-D hyperbolic solvers with [Muscl](#) scheme.*

 - class [Muscl3DT](#)
- Class for 3-D hyperbolic solvers with [Muscl](#) scheme using tetrahedra.*

 - class [BiotSavart](#)
- Class to compute the magnetic induction from the current density using the Biot-Savart formula.*

 - class [EC2D1T3](#)
- Eddy current problems in 2-D domains using solenoidal approximation.*

 - class [Equa_Electromagnetics](#)
- Abstract class for Electromagnetics [Equation](#) classes.*

 - class [HelmholtzBT3](#)
- Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.*

 - class [Equation](#)
- Abstract class for all equation classes.*

 - class [Equa_Fluid](#)
- Abstract class for Fluid Dynamics [Equation](#) classes.*

 - class [NSP2DQ41](#)
- Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using Q_1/P_0 element and a penalty formulation for the incompressibility condition.*

 - class [TINS2DT3B](#)
- Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.*

 - class [FastMarching2D](#)
- To run a Fast Marching Method on 2-D structured uniform grids.*

 - class [FMM2D](#)
- class for the fast marching 2-D algorithm*

 - class [FMM3D](#)
- class for the 3-D fast marching algorithm*

 - class [FMMSolver](#)
- The Fast Marching Method solver.*

 - class [Equa_Laplace](#)
- Abstract class for classes about the Laplace equation.*

 - class [Laplace1DL2](#)
- To build element equation for a 1-D elliptic equation using the 2-Node line element (P_1).*

 - class [Laplace1DL3](#)
- To build element equation for the 1-D elliptic equation using the 3-Node line (P_2).*

 - class [Laplace2DFVT](#)
- To build and solve the Laplace equation using a standard Finite Volume method.*

 - class [Laplace2DMHRT0](#)
- To build element equation for the 2-D elliptic equation using the Mixed Hybrid finite element at lowest degree (Raviart-Thomas RT_0).*

 - class [Laplace2DT3](#)
- To build element equation for the Laplace equation using the 2-D triangle element (P_1).*

 - class [SteklovPoincare2DBE](#)
- Solver of the Steklov Poincare problem in 2-D geometries using piecewise constant boundary elemen.*

 - class [Bar2DL2](#)

- To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node.*

 - class [Beam3DL2](#)
- To build element equations for 3-D beam equations using 2-node lines.*

 - class [Elas2DQ4](#)
- To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.*

 - class [Elas2DT3](#)
- To build element equations for 2-D linearized elasticity using 3-node triangles.*

 - class [Elas3DH8](#)
- To build element equations for 3-D linearized elasticity using 8-node hexahedra.*

 - class [Elas3DT4](#)
- To build element equations for 3-D linearized elasticity using 4-node tetrahedra.*

 - class [Equa.Solid](#)
- Abstract class for Solid Mechanics Finite Element classes.*

 - class [DC1DL2](#)
- Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.*

 - class [DC2DT3](#)
- Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.*

 - class [DC2DT6](#)
- Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.*

 - class [DC3DAT3](#)
- Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.*

 - class [DC3DT4](#)
- Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.*

 - class [Equa.Therm](#)
- Abstract class for Heat transfer Finite Element classes.*

 - class [PhaseChange](#)
- This class enables defining phase change laws for a given material.*

 - class [Funct](#)
- A simple class to parse real valued functions.*

 - class [IOField](#)
- Enables working with files in the XML Format.*

 - class [IPF](#)
- To read project parameters from a file in [IPF](#) format.*

 - class [Prescription](#)
- To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.*

 - class [Tabulation](#)
- To read and manipulate tabulated functions.*

 - class [UserData](#)
- Abstract class to define by user various problem data.*

 - class [BMatrix](#)
- To handle band matrices.*

- class [DMatrix](#)
To handle dense matrices.
- class [DSMatrix](#)
To handle symmetric dense matrices.
- class [LocalMatrix](#)
Handles small size matrices like element matrices, with a priori known size.
- class [Matrix](#)
Virtual class to handle matrices for all storage formats.
- class [Point](#)
Defines a point with arbitrary type coordinates.
- class [Point2D](#)
Defines a 2-D point with arbitrary type coordinates.
- class [Prec](#)
To set a preconditioner.
- class [TrMatrix](#)
To handle tridiagonal matrices.
- class [Domain](#)
To store and treat finite element geometric information.
- class [Edge](#)
To describe an edge.
- class [Element](#)
To store and treat finite element geometric information.
- class [Figure](#)
To store and treat a figure (or shape) information.
- class [Rectangle](#)
To store and treat a rectangular figure.
- class [Brick](#)
To store and treat a brick (parallelepiped) figure.
- class [Circle](#)
To store and treat a circular figure.
- class [Sphere](#)
To store and treat a sphere.
- class [Ellipse](#)
To store and treat an ellipsoidal figure.
- class [Triangle](#)
To store and treat a triangle.
- class [Polygon](#)
To store and treat a polygonal figure.
- class [Grid](#)
To manipulate structured grids.
- class [Material](#)
To treat material data. This class enables reading material data in material data files. It also returns these informations by means of its members.
- class [Mesh](#)
To store and manipulate finite element meshes.
- class [MeshAdapt](#)
To adapt mesh in function of given solution.

- class [NodeList](#)
Class to construct a list of nodes having some common properties.
- class [ElementList](#)
Class to construct a list of elements having some common properties.
- class [SideList](#)
Class to construct a list of sides having some common properties.
- class [EdgeList](#)
Class to construct a list of edges having some common properties.
- class [Node](#)
To describe a node.
- class [Partition](#)
To partition a finite element mesh into balanced submeshes.
- class [Side](#)
To store and treat finite element sides (edges in 2-D or faces in 3-D)
- class [Estimator](#)
To calculate an a posteriori estimator of the solution.
- class [Reconstruction](#)
To perform various reconstruction operations.
- class [FEShape](#)
Parent class from which inherit all finite element shape classes.
- class [triangle](#)
Defines a triangle. The reference element is the rectangle triangle with two unit edges.
- class [Hexa8](#)
Defines a three-dimensional 8-node hexahedral finite element using Q_1 -isoparametric interpolation.
- class [Line2](#)
To describe a 2-Node planar line finite element.
- class [Line2H](#)
To describe a 2-Node Hermite planar line finite element.
- class [Line3](#)
To describe a 3-Node quadratic planar line finite element.
- class [Penta6](#)
Defines a 6-node pentahedral finite element using P_1 interpolation in local coordinates $(s.x, s.y)$ and Q_1 isoparametric interpolation in local coordinates $(s.x, s.z)$ and $(s.y, s.z)$.
- class [Quad4](#)
Defines a 4-node quadrilateral finite element using Q_1 isoparametric interpolation.
- class [Tetra4](#)
Defines a three-dimensional 4-node tetrahedral finite element using P_1 interpolation.
- class [Triang3](#)
Defines a 3-Node (P_1) triangle.
- class [Triang6S](#)
Defines a 6-Node straight triangular finite element using P_2 interpolation.
- class [Eigen](#)
Class to find the smallest eigenvalues and corresponding eigenvectors in a generalized eigenproblem using the Bathe subspace iteration method.
- class [Iter](#)
Class to drive an iterative process.
- class [Precond](#)

An abstract class from which derive all preconditioning classes.

- class [Gauss](#)

Calculate data for Gauss integration.

- class [Timer](#)

To handle elapsed time counting.

Enumerations

- enum [PDE_Terms](#) {
[MASS](#) = 1,
[LUMPED_MASS](#) = 2,
[CAPACITY](#) = 1,
[LUMPED_CAPACITY](#) = 2,
[VISCOSITY](#) = 4,
[STIFFNESS](#) = 4,
[DIFFUSION](#) = 4,
[CONVECTION](#) = 8,
[DEVIATORIC](#) = 8,
[DILATATION](#) = 16,
[ELECTRIC](#) = 8,
[MAGNETIC](#) = 16,
[LOAD](#) = 32,
[HEAT_SOURCE](#) = 32,
[BOUNDARY_TRACTION](#) = 64,
[HEAT_FLUX](#) = 64,
[CONTACT](#) = 128,
[BUOYANCY](#) = 16,
[LORENTZ_FORCE](#) = 32 }
- enum [EqDataType](#) {
[INITIAL_FIELD](#) = 1,
[SOLUTION](#) = 1,
[INITIAL_AUX_1](#) = 2,
[INITIAL_AUX_2](#) = 3,
[INITIAL_AUX_3](#) = 4,
[INITIAL_AUX_4](#) = 5,
[BOUNDARY_CONDITION](#) = 6,
[BODY_FORCE](#) = 7,
[SOURCE](#) = 7,
[POINT_FORCE](#) = 8,
[BOUNDARY_FORCE](#) = 9,
[FLUX](#) = 9,
[TRACTION](#) = 9,
[AUX_INPUT_FIELD_1](#) = 10,
[AUX_INPUT_FIELD_2](#) = 11,
[AUX_INPUT_FIELD_3](#) = 12,
[AUX_INPUT_FIELD_4](#) = 13,
[DISPLACEMENT_FIELD](#) = 14,
[VELOCITY_FIELD](#) = 15,
[TEMPERATURE_FIELD](#) = 16 }
- enum [ArrayType](#) {
[LOCAL_ARRAY](#) = 0,
[GLOBAL_ARRAY](#) = 1 }

- enum `TimeScheme` {
`STATIONARY` = 0,
`FORWARD_EULER` = 1,
`BACKWARD_EULER` = 2,
`CRANK_NICOLSON` = 3,
`HEUN` = 4,
`NEWMARK` = 5,
`LEAP_FROG` = 6,
`ADAMS_BASHFORTH` = 7,
`AB2` = 7,
`RUNGE_KUTTA` = 8,
`RK4` = 8 }
 - enum `PDE` {
`LAPLACE`,
`DIFFUSION_CONVECTION`,
`THERMAL_PHASE_CHANGE`,
`INCOMPRESSIBLE_NAVIER_STOKES`,
`LINEARIZED_ELASTICITY`,
`PLANAR_TRUSS`,
`SPATIAL_BEAM` }
 - enum `FEType` {
`FE_2D_3N`,
`FE_2D_6N`,
`FE_2D_4N`,
`FE_3D_AXI_3N`,
`FE_3D_4N`,
`FE_3D_8N` }
 - enum `AnalysisType` {
`STEADY_STATE`,
`TRANSIENT`,
`OPTIMIZATION` }
 - enum `MatrixType` {
`SKYLINE` = 0x00800000,
`SPARSE` = 0x01000000,
`DIAGONAL` = 0x02000000,
`TRIDIAGONAL` = 0x04000000,
`SYMMETRIC` = 0x08000000,
`UNSYMMETRIC` = 0x10000000,
`IDENTITY` = 0x12000000 }
 - enum `Iteration` {
`DIRECT_SOLVER` = 0,
`CG_SOLVER` = 1,
`CGS_SOLVER` = 2,
`BICG_SOLVER` = 3,
`BICG_STAB_SOLVER` = 4,
`GMRES_SOLVER` = 5,
`QMR_SOLVER` = 6 }
- Choose iterative solver for the linear system.*
- enum `Preconditioner` {
`IDENT_PREC` = 0,
`DIAG_PREC` = 1,
`ILU_PREC` = 2,
`SSOR_PREC` = 3 }

Choose preconditioner for the linear system.

- enum `BCType` {
`PERIODIC_A` = 9999,
`PERIODIC_B` = -9999,
`CONTACT_BC` = 9998 ,
`SLIP` = 9996 }

Functions

- `ostream & operator<<` (`ostream &s`, const `Muscl3DT` &m)
Output mesh data as calculated in class `Muscl3DT`.
- `T_ * A` ()
Return element matrix.
- `T_ * b` ()
Return element right-hand side.
- `T_ * Prev` ()
Return element previous vector.
- `ostream & operator<<` (`ostream &s`, const `std::complex< double >` &x)
Output a complex number.
- `ostream & operator<<` (`ostream &s`, const `std::string` &c)
Output a string.
- `template<class T_ >`
`ostream & operator<<` (`ostream &s`, const `vector< T_ >` &v)
Output a vector instance.
- `template<class T_ >`
`ostream & operator<<` (`ostream &s`, const `std::pair< T_, T_ >` &a)
Output a pair instance.
- `void saveField` (`Vect< real_t >` &v, `string` output_file, `int` opt)
Save a vector to an output file in a given file format.
- `void saveField` (`Vect< real_t >` &v, const `Grid` &g, `string` output_file, `int` opt)
Save a vector to an output file in a given file format, for a structured grid data.
- `void saveGnuplot` (`string` input_file, `string` output_file, `string` mesh_file)
Save a vector to an input `Gnuplot` file.
- `void saveTecplot` (`string` input_file, `string` output_file, `string` mesh_file)
Save a vector to an output file to an input `Tecplot` file.
- `void saveVTK` (`string` input_file, `string` output_file, `string` mesh_file)
Save a vector to an output `VTK` file.
- `void saveGmsh` (`string` input_file, `string` output_file, `string` mesh_file)
Save a vector to an output `Gmsh` file.
- `ostream & operator<<` (`ostream &s`, const `Tabulation` &t)
Output Tabulated function data.
- `template<class T_ , size_t N_, class E_ >`
`void element_assembly` (const `E_` &e, const `LocalVect< T_, N_ >` &be, `Vect< T_ >` &b)
Assemble local vector into global vector.
- `template<class T_ , size_t N_, class E_ >`
`void element_assembly` (const `E_` &e, const `LocalMatrix< T_, N_, N_ >` &ae, `Vect< T_ >` &b)
Assemble diagonal local vector into global vector.
- `template<class T_ , size_t N_, class E_ >`
`void element_assembly` (const `E_` &e, const `LocalMatrix< T_, N_, N_ >` &ae, `Matrix< T_ >` *A)

Assemble local matrix into global matrix.

- `template<class T_, size_t N_, class E_>`
`void element_assembly (const E_ &e, const LocalMatrix< T_, N_, N_ > &ae, SkMatrix< T_ > &A)`

Assemble local matrix into global skyline matrix.

- `template<class T_, size_t N_, class E_>`
`void element_assembly (const E_ &e, const LocalMatrix< T_, N_, N_ > &ae, SkSMatrix< T_ > &A)`

Assemble local matrix into global symmetric skyline matrix.

- `template<class T_, size_t N_, class E_>`
`void element_assembly (const E_ &e, const LocalMatrix< T_, N_, N_ > &ae, SpMatrix< T_ > &A)`

Assemble local matrix into global sparse matrix.

- `template<class T_, size_t N_>`
`void side_assembly (const Element &e, const LocalMatrix< T_, N_, N_ > &ae, SpMatrix< T_ > &A)`

Side assembly of local matrix into global matrix (as instance of class SpMatrix).

- `template<class T_, size_t N_>`
`void side_assembly (const Element &e, const LocalMatrix< T_, N_, N_ > &ae, SkSMatrix< T_ > &A)`

Side assembly of local matrix into global matrix (as instance of class SkSMatrix).

- `template<class T_, size_t N_>`
`void side_assembly (const Element &e, const LocalMatrix< T_, N_, N_ > &ae, SkMatrix< T_ > &A)`

Side assembly of local matrix into global matrix (as instance of class SkMatrix).

- `template<class T_, size_t N_>`
`void side_assembly (const Element &e, const LocalVect< T_, N_ > &be, Vect< T_ > &b)`

Side assembly of local vector into global vector.

- `template<class T_>`
`BMatrix< T_ > operator* (T_ a, const BMatrix< T_ > &A)`

*Operator * (Premultiplication of matrix by constant)*

- `template<class T_>`
`ostream & operator<< (ostream &s, const BMatrix< T_ > &a)`

Output matrix in output stream.

- `template<class T_>`
`ostream & operator<< (ostream &s, const DMatrix< T_ > &a)`

Output matrix in output stream.

- `template<class T_>`
`ostream & operator<< (ostream &s, const DSMatrix< T_ > &a)`

Output matrix in output stream.

- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator* (T_ a, const LocalMatrix< T_, NR_, NC_ > &x)`

*Operator * (Multiply matrix x by scalar a)*

- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator/ (T_ a, const LocalMatrix< T_, NR_, NC_ > &x)`

Operator / (Divide matrix x by scalar a)

- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator+ (const LocalMatrix< T_, NR_, NC_ > &x, const LocalMatrix< T_, NR_, NC_ > &y)`

- Operator + (Add matrix x to y)*
- `template<class T_, size_t NR_, size_t NC_>`
`LocalMatrix< T_, NR_, NC_ > operator+ (const LocalMatrix< T_, NR_, NC_ > &x, const LocalMatrix< T_, NR_, NC_ > &y)`
Operator + (Subtract matrix y from x)
 - `template<class T_, size_t NR_, size_t NC_>`
`ostream & operator<< (ostream &s, const LocalMatrix< T_, NR_, NC_ > &a)`
Output vector in output stream.
 - `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator+ (const LocalVect< T_, N_ > &x, const LocalVect< T_, N_ > &y)`
Operator + (Add two vectors)
 - `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator- (const LocalVect< T_, N_ > &x, const LocalVect< T_, N_ > &y)`
Operator - (Subtract two vectors)
 - `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator* (T_ a, const LocalVect< T_, N_ > &x)`
*Operator * (Premultiplication of vector by constant)*
 - `template<class T_, size_t N_>`
`LocalVect< T_, N_ > operator/ (T_ a, const LocalVect< T_, N_ > &x)`
Operator / (Division of vector by constant)
 - `template<class T_, size_t N_>`
`double Dot (const LocalVect< T_, N_ > &a, const LocalVect< T_, N_ > &b)`
Calculate dot product of 2 vectors (instances of class LocalVect)
 - `template<class T_, size_t N_>`
`void Scale (T_ a, const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
Multiply vector x by constant a and store result in y.
 - `template<class T_, size_t N_>`
`void Scale (T_ a, LocalVect< T_, N_ > &x)`
Multiply vector x by constant a and store result in x.
 - `template<class T_, size_t N_>`
`void Axy (T_ a, const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
*Add a*x to vector y.*
 - `template<class T_, size_t N_>`
`void Copy (const LocalVect< T_, N_ > &x, LocalVect< T_, N_ > &y)`
Copy vector x into vector y.
 - `template<class T_, size_t N_>`
`ostream & operator<< (ostream &s, const LocalVect< T_, N_ > &v)`
Output vector in output stream.
 - `template<class T_ >`
`bool operator== (const Point< T_ > &a, const Point< T_ > &b)`
Operator ==
 - `template<class T_ >`
`Point< T_ > operator+ (const Point< T_ > &a, const Point< T_ > &b)`
Operator +
 - `template<class T_ >`
`Point< T_ > operator+ (const Point< T_ > &a, const T_ &x)`
Operator +

- `template<class T_ >`
`Point< T_ > operator-` (const `Point< T_ >` &a)
Unary Operator -
- `template<class T_ >`
`Point< T_ > operator-` (const `Point< T_ >` &a, const `Point< T_ >` &b)
Operator -
- `template<class T_ >`
`Point< T_ > operator-` (const `Point< T_ >` &a, const `T_` &x)
Operator -
- `template<class T_ >`
`Point< T_ > operator*` (const `T_` &a, const `Point< T_ >` &b)
*Operator **
- `template<class T_ >`
`Point< T_ > operator*` (const int &a, const `Point< T_ >` &b)
*Operator *.*
- `template<class T_ >`
`Point< T_ > operator*` (const `Point< T_ >` &b, const `T_` &a)
Operator /
- `template<class T_ >`
`Point< T_ > operator*` (const `Point< T_ >` &b, const int &a)
*Operator **
- `template<class T_ >`
`T_ operator*` (const `Point< T_ >` &a, const `Point< T_ >` &b)
*Operator **
- `template<class T_ >`
`Point< T_ > operator/` (const `Point< T_ >` &b, const `T_` &a)
Operator /
- `Point< double > CrossProduct` (const `Point< double >` &lp, const `Point< double >` &rp)
Return Cross product of two vectors lp and rp
- `bool areClose` (const `Point< double >` &a, const `Point< double >` &b, double toler=`OFELI.T-OLERANCE`)
Return true if both instances of class Point<double> are distant with less than toler
- `double SqrDistance` (const `Point< double >` &a, const `Point< double >` &b)
Return squared euclidean distance between points a and b
- `double Distance` (const `Point< double >` &a, const `Point< double >` &b)
Return euclidean distance between points a and b
- `template<class T_ >`
`std::ostream & operator<<` (std::ostream &s, const `Point< T_ >` &a)
Output point coordinates.
- `template<class T_ >`
`bool operator==` (const `Point2D< T_ >` &a, const `Point2D< T_ >` &b)
Operator ==.
- `template<class T_ >`
`Point2D< T_ > operator+` (const `Point2D< T_ >` &a, const `Point2D< T_ >` &b)
Operator +.
- `template<class T_ >`
`Point2D< T_ > operator+` (const `Point2D< T_ >` &a, const `T_` &x)
Operator +.

- `template<class T_ >`
`Point2D< T_ > operator-` (const `Point2D< T_ > &a`)
Unary Operator -
- `template<class T_ >`
`Point2D< T_ > operator-` (const `Point2D< T_ > &a`, const `Point2D< T_ > &b`)
Operator -
- `template<class T_ >`
`Point2D< T_ > operator-` (const `Point2D< T_ > &a`, const `T_ &x`)
Operator -
- `template<class T_ >`
`Point2D< T_ > operator*` (const `T_ &a`, const `Point2D< T_ > &b`)
*Operator *.*
- `template<class T_ >`
`Point2D< T_ > operator*` (const int `&a`, const `Point2D< T_ > &b`)
- `template<class T_ >`
`Point2D< T_ > operator*` (const `Point2D< T_ > &b`, const `T_ &a`)
Operator /
- `template<class T_ >`
`Point2D< T_ > operator*` (const `Point2D< T_ > &b`, const int `&a`)
*Operator **
- `template<class T_ >`
`T_ operator*` (const `Point2D< T_ > &b`, const `Point2D< T_ > &a`)
*Operator *.*
- `template<class T_ >`
`Point2D< T_ > operator/` (const `Point2D< T_ > &b`, const `T_ &a`)
Operator /
- `bool areClose` (const `Point2D< real_t > &a`, const `Point2D< real_t > &b`, `real_t` `toler=OFELI_TOLERANCE`)
Return true if both instances of class `Point2D<real_t>` are distant with less then toler [Default: `OFELI_EPSMCH`].
- `real_t SqrDistance` (const `Point2D< real_t > &a`, const `Point2D< real_t > &b`)
Return squared euclidean distance between points a and b
- `real_t Distance` (const `Point2D< real_t > &a`, const `Point2D< real_t > &b`)
Return euclidean distance between points a and b
- `template<class T_ >`
`std::ostream & operator<<` (std::ostream `&s`, const `Point2D< T_ > &a`)
Output point coordinates.
- `template<class T_ >`
`ostream & operator<<` (ostream `&s`, const `SkMatrix< T_ > &a`)
Output matrix in output stream.
- `template<class T_ >`
`ostream & operator<<` (ostream `&s`, const `SkSMatrix< T_ > &a`)
Output matrix in output stream.
- `template<class T_ >`
`TrMatrix< T_ > operator*` (`T_ a`, const `TrMatrix< T_ > &A`)
*Operator * (Premultiplication of matrix by constant)*
- `template<class T_ >`
`ostream & operator<<` (ostream `&s`, const `TrMatrix< T_ > &a`)
Output matrix in output stream.

- string `itos` (int i)
Function to convert an integer to a string.
- string `dtos` (real_t d)
Function to convert a real to a string.
- ostream & `operator<<` (ostream &s, const `Edge` &ed)
Output edge data.
- ostream & `operator<<` (ostream &s, const `Element` &el)
Output element data.
- `Figure operator&&` (const `Figure` &f1, const `Figure` &f2)
Function to define a `Figure` instance as the intersection of two `Figure` instances.
- `Figure operator-` (const `Figure` &f1, const `Figure` &f2)
Function to define a `Figure` instance as the set subtraction of two `Figure` instances.
- void `getMesh` (string file, ExternalFileFormat form, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in an external file format.
- void `getBamg` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Bamg` format.
- void `getEasymesh` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Easymesh` format.
- void `getGambit` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Gambit` neutral format.
- void `getGmsh` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Gmsh` format.
- void `getMatlab` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a Matlab mesh data.
- void `getNetgen` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Netgen` format.
- void `getTetgen` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Tetgen` format.
- void `getTriangle` (string file, `Mesh` &mesh, size_t nb_dof=1)
Construct an instance of class `Mesh` from a mesh file stored in `Triangle` format.
- ostream & `operator<<` (ostream &s, const `Grid` &g)
Output grid data.
- ostream & `operator<<` (ostream &s, const `Material` &m)
Output material data.
- ostream & `operator<<` (ostream &s, const `Mesh` &ms)
Output mesh data.
- ostream & `operator<<` (ostream &s, const `MeshAdapt` &a)
Output `MeshAdapt` class data.
- ostream & `operator<<` (ostream &s, const `NodeList` &nl)
Output `NodeList` instance.
- ostream & `operator<<` (ostream &s, const `ElementList` &el)
Output `ElementList` instance.
- ostream & `operator<<` (ostream &s, const `SideList` &sl)
Output `SideList` instance.
- ostream & `operator<<` (ostream &s, const `EdgeList` &el)
Output `EdgeList` instance.

- void `DeformMesh` (`Mesh` &mesh, const `Vect`< `real_t` > &u, `real_t` a=1)
Calculate deformed mesh using a displacement field.
- void `MeshToMesh` (`Mesh` &m1, `Mesh` &m2, const `Vect`< `real_t` > &u1, `Vect`< `real_t` > &u2, `size_t` nx, `size_t` ny=0, `size_t` nz=0, `size_t` dof=1)
Function to redefine a vector defined on a mesh to a new mesh.
- void `MeshToMesh` (`Mesh` &m1, `Mesh` &m2, const `Vect`< `real_t` > &u1, `Vect`< `real_t` > &u2, const `Point`< `real_t` > &xmin, const `Point`< `real_t` > &xmax, `size_t` nx, `size_t` ny, `size_t` nz, `size_t` dof=1)
Function to redefine a vector defined on a mesh to a new mesh.
- `real_t` `getMaxSize` (const `Mesh` &m)
Return maximal size of element edges for given mesh.
- `real_t` `getMinSize` (const `Mesh` &m)
Return minimal size of element edges for given mesh.
- `real_t` `getMinElementMeasure` (const `Mesh` &m)
Return minimal measure (length, area or volume) of elements of given mesh.
- `real_t` `getMinSideMeasure` (const `Mesh` &m)
Return minimal measure (length or area) of sides of given mesh.
- `real_t` `getMaxSideMeasure` (const `Mesh` &m)
Return maximal measure (length or area) of sides of given mesh.
- `real_t` `getMeanElementMeasure` (const `Mesh` &m)
Return average measure (length, area or volume) of elements of given mesh.
- `real_t` `getMeanSideMeasure` (const `Mesh` &m)
Return average measure (length or area) of sides of given mesh.
- void `setNodeCodes` (`Mesh` &m, const string &exp, int code, `size_t` dof=1)
Assign a given code to all nodes satisfying a boolean expression using node coordinates.
- void `setBoundaryNodeCodes` (`Mesh` &m, const string &exp, int code, `size_t` dof=1)
Assign a given code to all nodes on boundary that satisfy a boolean expression using node coordinates.
- void `setSideCodes` (`Mesh` &m, const string &exp, int code, `size_t` dof=1)
Assign a given code to all sides satisfying a boolean expression using node coordinates.
- void `setBoundarySideCodes` (`Mesh` &m, const string &exp, int code, `size_t` dof=1)
Assign a given code to all sides on boundary that satisfy a boolean expression using node coordinates.
- void `setElementCodes` (`Mesh` &m, const string &exp, int code)
Assign a given code to all elements satisfying a boolean expression using node coordinates.
- int `NodeInElement` (const `Node` *nd, const `Element` *el)
Say if a given node belongs to a given element.
- int `NodeInSide` (const `Node` *nd, const `Side` *sd)
Say if a given node belongs to a given side.
- int `SideInElement` (const `Side` *sd, const `Element` *el)
Say if a given side belongs to a given element.
- ostream & `operator<<` (ostream &s, const `Node` &nd)
Output node data.
- void `saveMesh` (const string &file, const `Mesh` &mesh, `ExternalFileFormat` form)
This function saves mesh data a file for a given external format.
- void `saveGmsh` (const string &gp_file, const `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `Gmsh` format.
- void `saveGnuplot` (const string &file, const `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `Gmsh` format.

- void `saveMatlab` (const string &file, const `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `Matlab` format.
- void `saveTecplot` (const string &file, const `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `Tecplot` format.
- void `saveVTK` (const string &file, const `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `VTK` format.
- void `saveBamg` (const string &file, `Mesh` &mesh)
This function outputs a `Mesh` instance in a file in `Bamg` format.
- ostream & `operator<<` (ostream &s, const `Side` &sd)
Output side data.
- ostream & `operator<<` (ostream &s, const `Estimator` &r)
Output estimator vector in output stream.
- template<class T_, class M_, class P_ >
int `BiCG` (const M_ &A, const P_ &P, const `Vect`< T_ > &b, `Vect`< T_ > &x, int max_it, double &toler, int verbose)
Biconjugate gradient solver function.
- template<class T_, class M_, class P_ >
int `BiCGStab` (const M_ &A, const P_ &P, const `Vect`< T_ > &b, `Vect`< T_ > &x, int max_it, double &toler, int verbose)
Biconjugate gradient stabilized solver function.
- void `BSpline` (size_t n, size_t t, `Vect`< `Point`< real_t > > &control, `Vect`< `Point`< real_t > > &output, size_t num_output)
Function to perform a B-spline interpolation.
- template<class T_, class M_, class P_ >
int `CG` (const M_ &A, const P_ &P, const `Vect`< T_ > &b, `Vect`< T_ > &x, int max_it, double &toler, int verbose)
Conjugate gradient solver function.
- template<class T_, class M_, class P_ >
int `CGS` (const M_ &A, const P_ &P, const `Vect`< T_ > &b, `Vect`< T_ > &x, int max_it, real_t &toler, int verbose)
Conjugate Gradient Squared solver function.
- template<class T_, class M_, class P_ >
int `GMRes` (const M_ &A, const P_ &P, const `Vect`< T_ > &b, `Vect`< T_ > &x, size_t m, int max_it, double &toler, int verbose)
GMRes solver function.
- template<class T_, class M_ >
int `GS` (const M_ &A, const `Vect`< T_ > &b, `Vect`< T_ > &x, real_t omega, int max_it, real_t &toler, int verbose)
Gauss-Seidel solver function.
- template<class T_, class M_ >
int `Jacobi` (const M_ &A, const `Vect`< T_ > &b, `Vect`< T_ > &x, real_t omega, int max_it, real_t toler, int verbose)
Jacobi solver function.
- void `BCAsConstraint` (const `Mesh` &m, const `Vect`< real_t > &bc, `Vect`< real_t > &up, `Vect`< real_t > &low)
*To impose Dirichlet boundary conditions in an optimization problem. If such conditions are to present, this function has to be invoked by giving on input `bc(i)` as the value to impose for the *i*-th optimization variable.*

- `template<class OPT_>`
`int OptimSA (OPT_ &theOpt, Vect< real_t > &x, real_t &rt, real_t &eps, int &ns, int &nt, int &neps, int &maxevl, Vect< real_t > &lb, Vect< real_t > &ub, Vect< real_t > &c, int &msg_lvl, int &seed1, int &seed2, real_t &t, Vect< real_t > &vm, Vect< real_t > &xopt, real_t &fopt, int &nacc, int &nfcnev, int &nobds)`
Simulated annealing optimization solver.
- `template<class OPT_>`
`int OptimTN (OPT_ &theOpt, Vect< real_t > &x, Vect< real_t > &low, Vect< real_t > &up, Vect< int > &pivot, int max_it, real_t toler, int msg_lvl)`
Truncated Newton optimization solver.
- `template<class M_ , class P1_ , class P2_ >`
`int QMR (const M_ &A, const P1_ &P1, const P2_ &P2, const Vect< real_t > &b, Vect< real_t > &x, int max_it, real_t &toler, int verbose)`
QMR solver function.
- `template<class T_ , class M_ >`
`int Richardson (const M_ &A, const Vect< T_ > &b, Vect< T_ > &x, real_t omega, int max_it, real_t toler, int verbose)`
Richardson solver function.
- `template<class T_ , class MAT_A_ , class MAT_D_ , class MAT_U_ >`
`void Schur (MAT_A_ &A, MAT_U_ &U, MAT_U_ &L, MAT_D_ &D, Vect< T_ > &b, Vect< T_ > &c)`
Solve a linear system of equations with a 2x2-block matrix.
- `template<class T_ , class MAT_A_ , class MAT_D_ , class MAT_U_ >`
`void Schur (Vect< MAT_A_ > &A, Vect< MAT_U_ > &U, Vect< MAT_U_ > &L, MAT_D_ &D, const Vect< Vect< T_ > > &b, const Vect< T_ > &c, Vect< Vect< T_ > > &x, Vect< T_ > &y)`
Solve a linear arrow block system by block factorization.
- `template<class T_ , class M_ >`
`int SSOR (const M_ &A, const Vect< T_ > &b, Vect< T_ > &x, int max_it, double toler, int verbose)`
SSOR solver function.
- `void banner (const string &prog=" ")`
Outputs a banner as header of any developed program.
- `template<class T_ >`
`void QuickSort (std::vector< T_ > &a, int begin, int end)`
Function to sort a vector.
- `template<class T_ >`
`void qksort (std::vector< T_ > &a, int begin, int end)`
Function to sort a vector.
- `template<class T_ , class C_ >`
`void qksort (std::vector< T_ > &a, int begin, int end, C_ compare)`
Function to sort a vector according to a key function.
- `int Sgn (real_t a)`
Return sign of a: -1 or 1.
- `real_t Abs2 (complex_t a)`
Return square of modulus of complex number a
- `real_t Abs2 (real_t a)`
Return square of real number a
- `real_t Abs (real_t a)`

- Return absolute value of a*

 - `real_t Abs (complex_t a)`
- Return modulus of complex number a*

 - `real_t Abs (const Point< real_t > &p)`
- Return Norm of vector a*

 - `real_t Conjg (real_t a)`
- Return complex conjugate of real number a*

 - `complex_t Conjg (complex_t a)`
- Return complex conjugate of complex number a*

 - `real_t Max (real_t a, real_t b, real_t c)`
- Return maximum value of real numbers a , b and c*

 - `int Max (int a, int b, int c)`
- Return maximum value of integer numbers a , b and c*

 - `real_t Min (real_t a, real_t b, real_t c)`
- Return minimum value of real numbers a , b and c*

 - `int Min (int a, int b, int c)`
- Return minimum value of integer numbers a , b and c*

 - `real_t Max (real_t a, real_t b, real_t c, real_t d)`
- Return maximum value of integer numbers a , b , c and d*

 - `int Max (int a, int b, int c, int d)`
- Return maximum value of integer numbers a , b , c and d*

 - `real_t Min (real_t a, real_t b, real_t c, real_t d)`
- Return minimum value of real numbers a , b , c and d*

 - `int Min (int a, int b, int c, int d)`
- Return minimum value of integer numbers a , b , c and d*

 - `real_t Arg (complex_t x)`
- Return argument of complex number x*

 - `complex_t Log (complex_t x)`
- Return principal determination of logarithm of complex number x*

 - `template<class T_ >`
`T_ Sqr (T_ x)`
Return square of value x
- `template<class T_ >`
`void Scale (T_ a, const vector< T_ > &x, vector< T_ > &y)`
Multiply vector x by a and save result in vector y
- `template<class T_ >`
`void Scale (T_ a, vector< T_ > &x)`
Multiply vector x by a
- `template<class T_ >`
`void Xpy (size_t n, T_ *x, T_ *y)`
Add array x to y
- `template<class T_ >`
`void Xpy (const vector< T_ > &x, vector< T_ > &y)`
Add vector x to y
- `template<class T_ >`
`void Axy (size_t n, T_ a, T_ *x, T_ *y)`
Multiply array x by a and add result to y

- `template<class T_ >`
`void Axy (T_ a, const vector< T_ > &x, vector< T_ > &y)`
Multiply vector x by a and add result to y
- `template<class T_ >`
`void Copy (size_t n, T_ *x, T_ *y)`
Copy array x to y n is the arrays size.
- `real_t Error2 (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute L^2 error between vectors x and y
- `real_t RError2 (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute L^2 error between vectors x and y
- `real_t ErrorMax (const vector< real_t > &x, const vector< real_t > &y)`
Return absolute Max. error between vectors x and y
- `real_t RErrorMax (const vector< real_t > &x, const vector< real_t > &y)`
Return relative Max. error between vectors x and y
- `template<class T_ >`
`T_ Dot (size_t n, T_ *x, T_ *y)`
Return dot product of arrays x and y
- `template<class T_ >`
`T_ Dot (const vector< T_ > &x, const vector< T_ > &y)`
Return dot product of vectors x and y .
- `real_t Dot (const vector< real_t > &x, const vector< real_t > &y)`
Return dot product of vectors x and y
- `template<class T_ >`
`T_ Dot (const Point< T_ > &x, const Point< T_ > &y)`
Return dot product of x and y
- `template<class T_ >`
`void Clear (vector< T_ > &v)`
Assign 0 to all entries of a vector.
- `real_t Nrm2 (size_t n, real_t *x)`
Return 2-norm of array x
- `real_t Nrm2 (const vector< real_t > &x)`
Return 2-norm of vector x
- `template<class T_ >`
`real_t Nrm2 (const Point< T_ > &a)`
Return 2-norm of a
- `bool Equal (real_t x, real_t y, real_t toler=OFELI_EPSMCH)`
Function to return true if numbers x and y are close up to a given tolerance $toler$
- `char itoc (int i)`
Function to convert an integer to a character.
- `template<class T_ >`
`T_ stringTo (const std::string &s)`
Function to convert a string to a template type parameter.
- `void RTrim (char *s)`
Function to remove blanks at the end of a string.
- `void LTrim (char *s)`
Function to remove blanks at the beginning of a string.
- `void Trim (char *s)`

Function to remove blanks at the beginning and end of a string.

- `template<class T_ >`
`void Swap (T_ &a, T_ &b)`
Swap elements a and b.

Variables

- `Node * theNode`
A pointer to [Node](#).
- `Element * theElement`
A pointer to [Element](#).
- `Side * theSide`
A pointer to [Side](#).
- `Edge * theEdge`
A pointer to [Edge](#).
- `unsigned int theStep`
Time step counter.
- `unsigned int theIteration`
Iteration counter.
- `unsigned int NbTimeSteps`
Number of time steps.
- `unsigned int MaxNbIterations`
Maximal number of iterations.
- `unsigned int Verbosity`
Parameter for verbosity of message outputting.
- `double theTimeStep`
Time step label.
- `double theTime`
Time value.
- `double theFinalTime`
Final time value.
- `double theTolerance`
Tolerance value for convergence.
- `double theDiscrepancy`
Value of discrepancy for an iterative procedure Its default value is 1.0.
- `bool Converged`
Boolean variable to say if an iterative procedure has converged.
- `bool InitPetsc`

6.1.1 Detailed Description

A namespace to group all library classes, functions, ... Namespace [OFELI](#) groups all OFELI library classes, functions and global variables.

6.1.2 Enumeration Type Documentation

enum PDE_Terms

Enumerate variable that selects various terms in partial differential equations

Enumerator

MASS Consistent mass term
LUMPED_MASS Lumped mass term
CAPACITY Consistent capacity term
LUMPED_CAPACITY Lumped capacity term
VISCOSITY Viscosity term
STIFFNESS Stiffness term
DIFFUSION Diffusion term
CONVECTION Convection term
DEVIATORIC Deviatoric term
DILATATION Dilatational term
ELECTRIC Electric term
MAGNETIC Magnetic term
LOAD Body load term
HEAT_SOURCE Body heat source term
BOUNDARY_TRACTION Boundary traction (pressure) term
HEAT_FLUX Boundary heat flux term
CONTACT Signorini contact
BUOYANCY Buoyancy force term
LORENTZ_FORCE Lorentz force term

enum EqDataType

Enumerate variable that selects equation data type

Enumerator

INITIAL_FIELD Initial condition
SOLUTION Solution vector (same as Initial)
INITIAL_AUX_1 Initial auxiliary field
INITIAL_AUX_2 Initial auxiliary field
INITIAL_AUX_3 Initial auxiliary field
INITIAL_AUX_4 Initial auxiliary field
BOUNDARY_CONDITION Boundary condition data
BODY_FORCE Body force data
SOURCE Source data (same as Body force)
POINT_FORCE Localized (at point) force
BOUNDARY_FORCE Boundary force data
FLUX Flux data (same as Boundary force)
TRACTION Traction data (same as Boundary force)

AUX_INPUT_FIELD_1 Auxiliary input field 1
AUX_INPUT_FIELD_2 Auxiliary input field 2
AUX_INPUT_FIELD_3 Auxiliary input field 3
AUX_INPUT_FIELD_4 Auxiliary input field 4
DISPLACEMENT_FIELD A displacement field
VELOCITY_FIELD A velocity field
TEMPERATURE_FIELD A temperature field

enum ArrayType

Selects local or global option for array as argument.

Enumerator

LOCAL_ARRAY For a local array labeled with local numbering
GLOBAL_ARRAY For a local array labeled with global numbering

enum TimeScheme

Selects time integration scheme

Enumerator

STATIONARY No time scheme: stationary
FORWARD_EULER Forward Euler scheme (Explicit)
BACKWARD_EULER Backward Euler scheme (Implicit)
CRANK_NICOLSON Crank-Nicolson scheme
HEUN Heun scheme
NEWMARK Newmark scheme
LEAP_FROG Leap Frog scheme
ADAMS_BASHFORTH Adams-Bashforth scheme (2nd Order)
AB2 Adams-Bashforth scheme (2nd Order)
RUNGE_KUTTA 4-th Order Runge-Kutta scheme (4th Order)
RK4 4-th Order Runge-Kutta scheme

enum PDE

Choose partial differential equation to ‘

Enumerator

LAPLACE Laplace equation
DIFFUSION_CONVECTION Diffusion Convection equation
THERMAL_PHASE_CHANGE Thermal phase change problem (Stefan)
INCOMPRESSIBLE_NAVIER_STOKES Incompressible Navier-Stokes equations
LINEARIZED_ELASTICITY Linearized elasticity equations
PLANAR_TRUSS 2-D truss equation
SPATIAL_BEAM 3-D beam equations

enum FEType

Choose Finite [Element](#) Type

Enumerator

FE_2D_3N 2-D elements, 3-Nodes (P1)
FE_2D_6N 2-D elements, 6-Nodes (P2)
FE_2D_4N 2-D elements, 4-Nodes (Q1)
FE_3D_AXI_3N 3-D Axisymmetric elements, 3-Nodes (P1)
FE_3D_4N 3-D elements, 4-Nodes (P1)
FE_3D_8N 3-D elements, 8-Nodes (Q1)

enum AnalysisType

Choose analysis type

Enumerator

STEADY_STATE Steady state analysis
TRANSIENT Transient analysis
OPTIMIZATION Optimization analysis

enum MatrixType

Choose matrix storage and type

Enumerator

SKYLINE Skyline storage
SPARSE Sparse storage
DIAGONAL Diagonal storage
TRIDIAGONAL Tridiagonal storage
SYMMETRIC Symmetric matrix
UNSYMMETRIC Unsymmetric matrix
IDENTITY Identity matrix

enum Iteration

Choose iterative solver for the linear system.

Enumerator

DIRECT_SOLVER Direct solver
CG_SOLVER CG Method
CGS_SOLVER CGS Method
BICG_SOLVER BiCG Method
BICG_STAB_SOLVER BiCGStab Method
GMRES_SOLVER GMRes Method
QMR_SOLVER QMR Method

enum Preconditioner

Choose preconditioner for the linear system.

Enumerator

- IDENT_PREC** Identity (No preconditioning)
- DIAG_PREC** Diagonal preconditioner
- ILU_PREC** ILU (Incomplete factorization) preconditioner
- SSOR_PREC** SSOR preconditioner

enum BCType

To select special boundary conditions.

Enumerator

- PERIODIC_A** Periodic Boundary conditions (first side)
- PERIODIC_B** Periodic Boundary conditions (second side)
- CONTACT_BC** Contact Boundary conditions
- SLIP** Slip Boundary conditions

6.1.3 Function Documentation**T_* OFELI::A ()**

Return element matrix.

Matrix is returned as a C-array

T_* OFELI::b ()

Return element right-hand side.

Right-hand side is returned as a C-array

T_* OFELI::Prev ()

Return element previous vector.

This is the vector given in time dependent constructor. It is returned as a C-array.

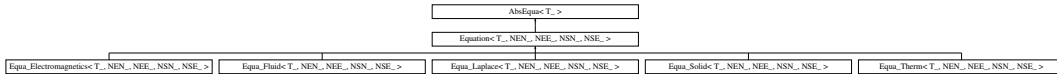
Chapter 7

Class Documentation

7.1 AbsEqua< T_ > Class Template Reference

Mother abstract class to describe equation.

Inheritance diagram for AbsEqua< T_ >:



Public Member Functions

- [AbsEqua](#) ()
Default constructor.
- [AbsEqua](#) (class [Mesh](#) &mesh)
Constructor with mesh instance.
- virtual [~AbsEqua](#) ()
Destructor.
- void [setMesh](#) (class [Mesh](#) &m)
Define mesh and renumber DOFs after removing imposed ones.
- [Mesh](#) & [getMesh](#) () const
Return reference to [Mesh](#) instance.
- [LinearSolver](#)< T_ > & [getLinearSolver](#) ()
Return reference to linear solver instance.
- void [setSolver](#) (int ls, int pc=[IDENT_PREC](#))
Choose solver for the linear system.
- int [solveEigenProblem](#) (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- [real.t](#) [getEigenValue](#) (int n) const
Return the n-th eigenvalue.
- void [getEigenvector](#) (int n, [Vect](#)< [real.t](#) > &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class [Eigen](#) & [getEigenSolver](#) ()
Return reference to eigenproblem solver.

7.1.1 Detailed Description

template<class T_>class OFELI::AbsEqua< T_ >

Mother abstract class to describe equation.

Template Parameters

< T_ >	Data type (real_t, float, complex<real_t>, ...)
--------	---

7.1.2 Member Function Documentation

Mesh& getMesh () const

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC)

Choose solver for the linear system.

Parameters

in	ls	<div>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</div> <ul style="list-style-type: none">• DIRECT_SOLVER, Use a facorization solver [default]• CG_SOLVER, Conjugate Gradient iterative solver• CGS_SOLVER, Squared Conjugate Gradient iterative solver• BICG_SOLVER, BiConjugate Gradient iterative solver• BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver• GMRES_SOLVER, GMRES iterative solver• QMR_SOLVER, QMR iterative solver
----	----	---

in	pc	Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values: <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
-----------	-----------	---

int solveEigenProblem (int nb_eigv, bool g = false)

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real.t > & v) const

Store the eigenvector corresponding to a given eigenvalue.

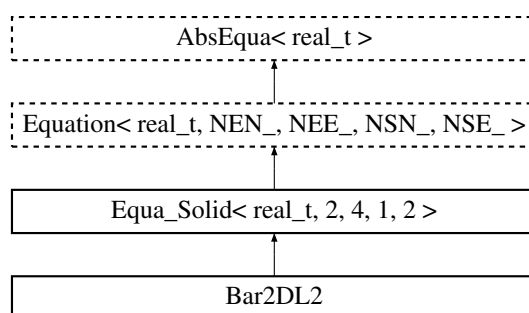
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.2 Bar2DL2 Class Reference

To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node.

Inheritance diagram for Bar2DL2:



Public Member Functions

- `Bar2DL2 ()`
Default Constructor.
- `Bar2DL2 (Element *el, real_t section)`
Constructor using element data.
- `~Bar2DL2 ()`
Destructor.
- `void Mass (real_t coef=1.)`
Add element consistent mass contribution to matrix and right-hand side after multiplication by coef
- `void LMass (real_t coef=1.)`
Add element lumped mass contribution to matrix and right-hand side after multiplication by coef
- `void LMassToLHS (real_t coef=1)`
Add lumped mass matrix to left-hand side after multiplying it by coefficient coef
- `void LMassToRHS (real_t coef=1)`
Add lumped mass contribution to right-hand side after multiplying it by coefficient coef
- `void MassToLHS (real_t coef=1)`
Add consistent mass matrix to left-hand side after multiplying it by coefficient coef
- `void MassToRHS (real_t coef=1)`
Add consistent mass contribution to right-hand side after multiplying it by coefficient coef
- `void Stiffness (real_t coef=1.)`
Add element stiffness to left hand side.
- `void BodyRHS (UserData< real_t > &ud)`
Add body right-hand side term to right hand side.
- `real_t Stress () const`
Return stresses in bar.
- `void getStresses (const Vect< real_t > &u, Vect< real_t > &s)`
Return stresses in the truss structure (elementwise)
- `void buildEigen (SkSMatrix< real_t > &K, SkSMatrix< real_t > &M)`
Build global stiffness and mass matrices for the eigen system.
- `void buildEigen (SkSMatrix< real_t > &K, Vect< real_t > &M)`
Build global stiffness and mass matrices for the eigen system.
- `void setLumpedMass ()`
Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.
- `void setMass ()`

Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.

- virtual void `Deviator` (`real_t` coef=1)
Add deviator matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `Dilatation` (`real_t` coef=1)
Add dilatation matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `DilatationToRHS` (`real_t` coef=1)
Add dilatation vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `DeviatorToRHS` (`real_t` coef=1)
Add deviator vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `StiffnessToRHS` (`real_t` coef=1)
Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- void `setDilatation` ()
Add dilatation matrix to left and/or right-hand side taking into account time.
- void `setDeviator` ()
Add deviator matrix to left and/or right-hand side taking into account time integration scheme.
- void `setStiffness` ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- void `updateBC` (const `Element` &el, const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector` (`Vect`< `real_t` > &b)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect`< `real_t` > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect`< `real_t` > &b)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.

- void `ElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< real_t > &v`)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix< real_t > *A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix< real_t > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< real_t > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix< real_t > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect< real_t > &v`)
Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element` &el, const `Vect< real_t > &x`, `Vect< real_t > &b`)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect< real_t > &x`, `Vect< real_t > &b`)
Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes` () const
Return number of element nodes.
- size_t `getNbEq` () const
Return number of element equations.
- `real_t * A` ()
Return element matrix as a C-array.
- `real_t * sA` ()
Return side matrix as a C-array.
- `real_t * b` ()
Return element right-hand side as a C-array.

- `real_t * sb ()`
Return side right-hand side as a C-array.
- `real_t * Prev ()`
Return element matrix as a C-array.
- `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
Return element matrix as a LocalMatrix instance.
- `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
Return side matrix as a LocalMatrix instance.
- `LocalVect< real_t, NEE_ > & Eb ()`
Return element right-hand side as a LocalVect instance.
- `LocalVect< real_t, NEE_ > & Ep ()`
Return element matrix as a C-array.
- `void setInitialSolution (const Vect< real_t > &u)`
Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`
Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`
Return reference to Mesh instance.
- `LinearSolver< real_t > & getLinearSolver ()`
Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`
Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n-th eigenvalue.
- `void getEigenVector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void Young (const real_t &E)`
Set (constant) Young modulus.
- `void Young (const string &exp)`
Set Young modulus given by an algebraic expression.
- `void Poisson (const real_t &nu)`
Set (constant) Poisson ratio.
- `void Poisson (const string &exp)`
Set Poisson ratio given by an algebraic expression.
- `void Density (const real_t &rho)`
Set (constant) density.
- `void Density (const string &exp)`
Set density given by an algebraic expression.

- void [setMaterial](#) ()
Set material properties.
- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.2.1 Detailed Description

To build element equations for Planar Elastic Bar element with 2 DOF (Degrees of Freedom) per node.

This class implements a planar (two-dimensional) elastic bar using 2-node lines. Note that members calculating element arrays have as an argument a real `coef` that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.2.2 Constructor & Destructor Documentation

Bar2DL2 ()

Default Constructor.

Constructs an empty equation.

Bar2DL2 ([Element](#) * el, [real_t](#) section)

Constructor using element data.

Parameters

in	el	Pointer to Element
in	section	Section of bar at present element

7.2.3 Member Function Documentation

void [Mass](#) ([real_t](#) coef = 1.) [virtual]

Add element consistent mass contribution to matrix and right-hand side after multiplication by `coef`

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
--------------------	----------------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void [LMass](#) ([real_t](#) coef = 1.) [virtual]

Add element lumped mass contribution to matrix and right-hand side after multiplication by `coef`

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
--------------------	----------------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void [LMassToLHS](#) ([real_t](#) coef = 1) [virtual]

Add lumped mass matrix to left-hand side after multiplying it by coefficient `coef`

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void LMassToRHS (real_t coef = 1) [virtual]

Add lumped mass contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void MassToLHS (real_t coef = 1) [virtual]

Add consistent mass matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void MassToRHS (real_t coef = 1) [virtual]

Add consistent mass contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void Stiffness (real_t coef = 1.) [virtual]

Add element stiffness to left hand side.

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 2, 4, 1, 2 >](#).

void BodyRHS (UserData< real_t > & ud)

Add body right-hand side term to right hand side.

Parameters

in	<i>ud</i>	instance containing user data with prescribes loads
-----------	-----------	---

void getStresses (const Vect< real_t > & u, Vect< real_t > & s)

Return stresses in the truss structure (elementwise)

Parameters

in	<i>u</i>	Vect instance containing displacements at nodes
in	<i>s</i>	Vect instance containing axial stresses in elements

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	K	Stiffness matrix
in	M	Vector containing diagonal mass matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
-----------	------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
-----------	-------------	--

<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <i>dof</i> is handled in the system
-----------	------------	---

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <i>ePrev</i> . This member function is to be used if a constructor with Element was invoked.
-----------	----------	--

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.
<i>in</i>	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real_t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system The resulting local vector can be accessed by attribute <i>ePrev</i> .

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real_t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute <i>ePrev</i> .
----	----------	--

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer *_theSide*

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Global matrix stored as an <code>SkSMatrix</code> instance
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>in</code>	<code>A</code>	Global matrix stored as an <code>SkMatrix</code> instance
-----------------	----------------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	sd	Reference to Side instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
-----------	-----------	---

in	pc	Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values: <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
-----------	-----------	---

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real.t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

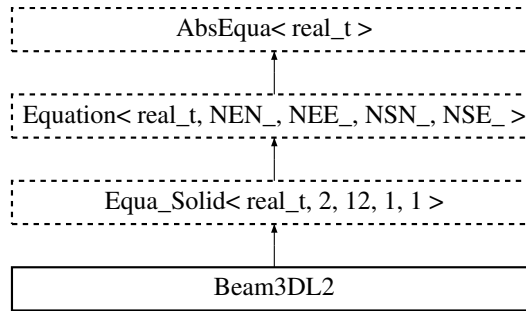
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.3 Beam3DL2 Class Reference

To build element equations for 3-D beam equations using 2-node lines.

Inheritance diagram for Beam3DL2:



Public Member Functions

- [Beam3DL2](#) ()
Default Constructor.
- [Beam3DL2](#) ([Element](#) *el, [real_t](#) A, [real_t](#) l1, [real_t](#) l2)
Constructor using element data.
- [Beam3DL2](#) ([Element](#) *el, [real_t](#) A, [real_t](#) l1, [real_t](#) l2, const [Vect](#)< [real_t](#) > &u, const [real_t](#) &time=0)
Constructor for dynamic problems.
- [Beam3DL2](#) ([Mesh](#) &ms, const [Vect](#)< [real_t](#) > &u, [Vect](#)< [real_t](#) > &d)
Constructor to determine displacements.
- [~Beam3DL2](#) ()
Destructor.
- void [LMassToLHS](#) ([real_t](#) coef=1.)
Add element lumped Mass contribution to matrix after multiplication by coef
- void [LMassToRHS](#) ([real_t](#) coef=1.)
Add element lumped Mass contribution to RHS after multiplication by coef
- void [MassToLHS](#) ([real_t](#) coef=1.)
Add element consistent Mass contribution to matrix after multiplication by coef (not implemented)
- void [MassToRHS](#) ([real_t](#) coef=1.)
Add element consistent Mass contribution to RHS after multiplication by coef (not implemented)
- void [Stiffness](#) ([real_t](#) coef=1.)
Add element stiffness to left hand side.
- void [Load](#) (const [Vect](#)< [real_t](#) > &f)
Add contributions for loads.
- void [setBending](#) ()
Set bending contribution to stiffness.
- void [setAxial](#) ()
Set axial contribution to stiffness.
- void [setShear](#) ()
Set shear contribution to stiffness.
- void [setTorsion](#) ()
Set torsion contribution to stiffness.
- void [setNoBending](#) ()
Set no bending contribution.
- void [setNoAxial](#) ()
Set no axial contribution.

- void `setNoShear ()`
Set no shear contribution.
- void `setNoTorsion ()`
Set no torsion contribution.
- void `setReducedIntegration ()`
Set reduced integration.
- `real_t AxialForce ()` const
Return axial force in element.
- `Point< real_t > ShearForce ()` const
Return shear force in element.
- `Point< real_t > BendingMoment ()` const
Return bending moment in element.
- `real_t TwistingMoment ()` const
Return twisting moment in element.
- void `buildEigen (SkSMatrix< real_t > &K, Vect< real_t > &M)`
Build global stiffness and mass matrices for the eigen system.
- void `setLumpedMass ()`
Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.
- void `setMass ()`
Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.
- virtual void `Mass (real_t coef=1)`
Add consistent mass matrix to left-hand side after multiplication by coef [Default: 1].
- virtual void `LMass (real_t coef=1)`
Add lumped mass matrix to left-hand side after multiplication by coef [Default: 1].
- virtual void `Deviator (real_t coef=1)`
Add deviator matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `Dilatation (real_t coef=1)`
Add dilatation matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `DilatationToRHS (real_t coef=1)`
Add dilatation vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `DeviatorToRHS (real_t coef=1)`
Add deviator vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void `StiffnessToRHS (real_t coef=1)`
Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- void `setDilatation ()`
Add dilatation matrix to left and/or right-hand side taking into account time.
- void `setDeviator ()`
Add deviator matrix to left and/or right-hand side taking into account time integration scheme.
- void `setStiffness ()`
Add convection contribution to left and/or right-hand side taking into account time integration scheme.

- void `buildEigen` (`SkSMatrix< real_t > &K`, `SkSMatrix< real_t > &M`)
Build global stiffness and mass matrices for the eigen system.
- void `updateBC` (const `Element` &el, const `Vect< real_t > &bc`)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect< real_t > &bc`)
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector` (`Vect< real_t > &b`)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect< real_t > &b`, `LocalVect< real_t, NEE_ > &be`)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect< real_t > &b`, `LocalVect< real_t, NEN_ > &be`, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect< real_t > &b`, `LocalVect< real_t, NEN_ > &be`)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect< real_t > &b`, `LocalVect< real_t, NSE_ > &be`)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect< real_t > &b`, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect< real_t > &b`)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< real_t > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< real_t > &v`)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.

- void `DGElementAssembly` (`TrMatrix`< `real_t` > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix`< `real_t` > *A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect`< `real_t` > &v)
Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of side matrix by side vector into global vector.
- `size_t` `getNbNodes` () const
Return number of element nodes.
- `size_t` `getNbEq` () const
Return number of element equations.
- `real_t` * `A` ()
Return element matrix as a C-array.
- `real_t` * `sA` ()
Return side matrix as a C-array.
- `real_t` * `b` ()
Return element right-hand side as a C-array.
- `real_t` * `sb` ()
Return side right-hand side as a C-array.
- `real_t` * `Prev` ()
Return element matrix as a C-array.
- `LocalMatrix`< `real_t`, `NEE_`, `NEE_` > & `EA` ()
Return element matrix as a LocalMatrix instance.
- `LocalMatrix`< `real_t`, `NSE_`, `NSE_` > & `SA` ()
Return side matrix as a LocalMatrix instance.
- `LocalVect`< `real_t`, `NEE_` > & `Eb` ()
Return element right-hand side as a LocalVect instance.
- `LocalVect`< `real_t`, `NEE_` > & `Ep` ()
Return element matrix as a C-array.
- void `setInitialSolution` (const `Vect`< `real_t` > &u)
Set initial solution (previous time step)
- `real_t` `setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.

- `LinearSolver< real_t > & getLinearSolver ()`
Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`
Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n-th eigenvalue.
- `void getEigenvector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void Young (const real_t &E)`
Set (constant) Young modulus.
- `void Young (const string &exp)`
Set Young modulus given by an algebraic expression.
- `void Poisson (const real_t &nu)`
Set (constant) Poisson ratio.
- `void Poisson (const string &exp)`
Set Poisson ratio given by an algebraic expression.
- `void Density (const real_t &rho)`
Set (constant) density.
- `void Density (const string &exp)`
Set density given by an algebraic expression.
- `void setMaterial ()`
Set material properties.
- `void Init (const Element *el)`
Set element arrays to zero.
- `void Init (const Side *sd)`
Set side arrays to zero.

7.3.1 Detailed Description

To build element equations for 3-D beam equations using 2-node lines.

This class enables building finite element arrays for 3-D beam elements using 6 degrees of freedom per node and 2-Node line elements.

7.3.2 Constructor & Destructor Documentation

Beam3DL2 (Element * el, real_t A, real_t I1, real_t I2)

Constructor using element data.

Parameters

in	<i>el</i>	Pointer to Element
in	<i>A</i>	Section area of the beam
in	<i>I1</i>	first (x) momentum of inertia
in	<i>I2</i>	second (y) momentum of inertia

Beam3DL2 ([Element](#) * *el*, *real_t* *A*, *real_t* *I1*, *real_t* *I2*, const [Vect](#)< *real_t* > & *u*, const *real_t* & *time* = 0)

Constructor for dynamic problems.

Parameters

in	<i>el</i>	Pointer to Element
in	<i>A</i>	Section area of the beam
in	<i>I1</i>	first (x) momentum of inertia
in	<i>I2</i>	second (y) momentum of inertia
in	<i>u</i>	Vector containing previous solution (at previous time step)
in	<i>time</i>	Current time value

Beam3DL2 ([Mesh](#) & *ms*, const [Vect](#)< *real_t* > & *u*, [Vect](#)< *real_t* > & *d*)

Constructor to determine displacements.

The unknowns consist in planar and rotational degrees of freedom. This member function construct a 3-D node vector that gives the displacement vector at each node.

Parameters

in	<i>ms</i>	Mesh instance
in	<i>u</i>	Vector containing the solution vector
out	<i>d</i>	Vector containing three components for each node that are x, y and z displacements.

7.3.3 Member Function Documentation

void buildEigen ([SkSMatrix](#)< *real_t* > & *K*, [Vect](#)< *real_t* > & *M*)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Vector containing diagonal mass matrix

void buildEigen ([SkSMatrix](#)< *real_t* > & *K*, [SkSMatrix](#)< *real_t* > & *M*) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<code>in</code>	<code>el</code>	Reference to current element instance
<code>in</code>	<code>bc</code>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<code>in</code>	<code>bc</code>	Vector that contains imposed values at all DOFs
-----------------	-----------------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<code>in</code>	<code>dof_type</code>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE_FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT_FIELD</code>, DOFs are supported by elements • <code>SIDE_FIELD</code>, DOFs are supported by sides
<code>in</code>	<code>dof</code>	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<code>in</code>	<code>b</code>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
-----------------	----------------	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

<code>in</code>	<code>b</code>	Global vector to be localized.
<code>out</code>	<code>be</code>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real_t > & b, int dof_type = *NODE_FIELD*, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF number <code>dof</code> is handled in the system The resulting local vector can be accessed by attribute <code>ePrev</code> .

Remarks

This member function is to be used if a constructor with `Element` was invoked. It uses the `Element` pointer `_theElement`

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
----	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

Parameters

in	<i>n</i>	Label of the eigenvalue
out	<i>v</i>	Vect instance containing the corresponding eigenvector. This vector is resized.

7.4 BiotSavart Class Reference

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

Public Member Functions

- [BiotSavart \(\)](#)
Default constructor.
- [BiotSavart \(Mesh &ms\)](#)
Constructor using mesh data.
- [BiotSavart \(Mesh &ms, const Vect< real_t > &J, Vect< real_t > &B, int code=0\)](#)
Constructor using mesh and vector of real current density.
- [BiotSavart \(Mesh &ms, const Vect< complex_t > &J, Vect< complex_t > &B, int code=0\)](#)
Constructor using mesh and vector of complex current density.
- [~BiotSavart \(\)](#)
Destructor.
- void [setCurrentDensity](#) (const Vect< real_t > &J)
Set (real) current density given at elements.
- void [setCurrentDensity](#) (const Vect< complex_t > &J)
Set (real) current density given at elements.
- void [setMagneticInduction](#) (Vect< real_t > &B)

- *Transmit (real) magnetic induction vector given at nodes.*
 • void `setMagneticInduction` (`Vect< complex_t > &B`)
- *Transmit (complex) magnetic induction vector given at nodes.*
 • void `selectCode` (`int code`)
- *Choose code of faces or edges at which current density is given.*
 • void `setPermeability` (`real_t mu`)
- *Set the magnetic permeability coefficient.*
 • void `setBoundary` ()
- *Choose to compute the magnetic induction at boundary nodes only.*
 • `Point< real_t > getB3` (`Point< real_t > x`)
- *Compute the real magnetic induction at a given point using the volume Biot-Savart formula.*
 • `Point< real_t > getB2` (`Point< real_t > x`)
- *Compute the real magnetic induction at a given point using the surface Biot-Savart formula.*
 • `Point< real_t > getB1` (`Point< real_t > x`)
- *Compute the real magnetic induction at a given point using the line Biot-Savart formula.*
 • `Point< complex_t > getBC3` (`Point< real_t > x`)
- *Compute the complex magnetic induction at a given point using the volume Biot-Savart formula.*
 • `Point< complex_t > getBC2` (`Point< real_t > x`)
- *Compute the complex magnetic induction at a given point using the surface Biot-Savart formula.*
 • `Point< complex_t > getBC1` (`Point< real_t > x`)
- *Compute the complex magnetic induction at a given point using the line Biot-Savart formula.*
 • `int run` ()
- *Run the calculation by the Biot-Savart formula.*

7.4.1 Detailed Description

Class to compute the magnetic induction from the current density using the Biot-Savart formula.

Given a current density vector given at elements, a collection of sides of edges (piecewise constant), this class enables computing the magnetic induction vector (continuous and piecewise linear) using the Ampere equation. This magnetic induction is obtained by using the Biot-Savart formula which can be either a volume, surface or line formula depending on the nature of the current density vector.

7.4.2 Constructor & Destructor Documentation

BiotSavart (Mesh & ms)

Constructor using mesh data.

Parameters

<code>in</code>	<code>ms</code>	<code>Mesh</code> instance
-----------------	-----------------	----------------------------

BiotSavart (Mesh & ms, const Vect< real_t > & J, Vect< real_t > & B, int code = 0)

Constructor using mesh and vector of real current density.

The current density is assumed piecewise constant

Parameters

<code>in</code>	<code>ms</code>	<code>Mesh</code> instance
<code>in</code>	<code>J</code>	Sidewise vector of current density (<code>J</code> is a real valued vector), in the case of a surface supported current
<code>in</code>	<code>B</code>	Nodewise vector that contains, once the member function <code>run</code> is used, the magnetic induction
<code>in</code>	<code>code</code>	Only sides with given <code>code</code> support current [Default: 0]

BiotSavart (Mesh & ms, const Vect< complex_t > & J, Vect< complex_t > & B, int code = 0)

Constructor using mesh and vector of complex current density.

The current density is assumed piecewise constant

Parameters

in	<i>ms</i>	Mesh instance
in	<i>J</i>	Sidewise vector of current density (J is a complex valued vector), in the case of a surface supported current
in	<i>B</i>	Nodewise vector that contains, once the member function run is used, the magnetic induction
in	<i>code</i>	Only sides with given code support current [Default: 0]

7.4.3 Member Function Documentation

void setCurrentDensity (const Vect< real_t > & J)

Set (real) current density given at elements.

The current density is assumed piecewise constant and real valued. This function can be used in the case of the volume Biot-Savart formula.

Parameters

in	<i>J</i>	Current density vector (Vect instance) and real entries
----	----------	--

void setCurrentDensity (const Vect< complex_t > & J)

Set (real) current density given at elements.

The current density is assumed piecewise constant and complex valued. This function can be used in the case of the volume Biot-Savart formula.

Parameters

in	<i>J</i>	Current density vector (Vect instance) of complex entries
----	----------	--

void setMagneticInduction (Vect< real_t > & B)

Transmit (real) magnetic induction vector given at nodes.

Parameters

out	<i>B</i>	Magnetic induction vector (Vect instance) and real entries
-----	----------	---

void setMagneticInduction (Vect< complex_t > & B)

Transmit (complex) magnetic induction vector given at nodes.

Parameters

out	<i>B</i>	Magnetic induction vector (Vect instance) and complex entries
-----	----------	--

void setPermeability (real_t mu)

Set the magnetic permeability coefficient.

Parameters

<code>in</code>	<code>mu</code>	Magnetic permeability
-----------------	-----------------	-----------------------

void setBoundary ()

Choose to compute the magnetic induction at boundary nodes only.

By default the magnetic induction is computed (using the function `run`) at all mesh nodes

Note

This function has no effect for surface of line Biot-Savart formula

Point<real_t> getB3 (Point< real_t > x)

Compute the real magnetic induction at a given point using the volume Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

Point<real_t> getB2 (Point< real_t > x)

Compute the real magnetic induction at a given point using the surface Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

Point<real_t> getB1 (Point< real_t > x)

Compute the real magnetic induction at a given point using the line Biot-Savart formula.

This function computes a real valued magnetic induction for a real valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

Point<complex_t> getBC3 (Point< real_t > x)

Compute the complex magnetic induction at a given point using the volume Biot-Savart formula.

This function computes a complex valued magnetic induction for a complex valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

Point<complex_t> getBC2 (Point< real_t > x)

Compute the complex magnetic induction at a given point using the surface Biot-Savart formula.

This function computes a complex valued magnetic induction for a complex valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

Point<complex_t> getBC1 (Point< real_t > x)

Compute the complex magnetic induction at a given point using the line Biot-Savart formula.

This function computes a complex valued magnetic induction for a complex valued current density field

Parameters

<code>in</code>	<code>x</code>	Coordinates of point at which the magnetic induction is computed
-----------------	----------------	--

Returns

Value of the magnetic induction at `x`

int run ()

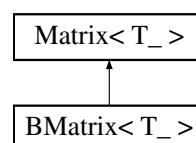
Run the calculation by the Biot-Savart formula.

This function computes the magnetic induction, which is stored in the vector `B` given in the constructor

7.5 BMatrix< T_ > Class Template Reference

To handle band matrices.

Inheritance diagram for `BMatrix< T_ >`:



Public Member Functions

- **BMatrix** ()
Default constructor.
- **BMatrix** (size_t **size**, int **ld**, int **ud**)
Constructor that for a band matrix with given size and bandwidth.
- **BMatrix** (const **BMatrix** &**m**)
Copy Constructor.
- **~BMatrix** ()
Destructor.
- void **setSize** (size_t **size**, int **ld**, int **ud**)
Set size (number of rows) and storage of matrix.
- void **MultAdd** (const **Vect**< T_ > &**x**, **Vect**< T_ > &**y**) const
Multiply matrix by vector x and add result to y
- void **MultAdd** (T_ **a**, const **Vect**< T_ > &**x**, **Vect**< T_ > &**y**) const
*Multiply matrix by vector $a*x$ and add result to y*
- void **Mult** (const **Vect**< T_ > &**x**, **Vect**< T_ > &**y**) const
Multiply matrix by vector x and save result in y
- void **TMult** (const **Vect**< T_ > &**x**, **Vect**< T_ > &**y**) const
Multiply transpose of matrix by vector x and save result in y
- void **Axpy** (T_ **a**, const **BMatrix**< T_ > &**x**)
Add to matrix the product of a matrix by a scalar.
- void **Axpy** (T_ **a**, const **Matrix**< T_ > ***x**)
Add to matrix the product of a matrix by a scalar.
- void **set** (size_t **i**, size_t **j**, const T_ &**val**)
Add constant val to an entry (i, j) of the matrix.
- void **add** (size_t **i**, size_t **j**, const T_ &**val**)
Add constant val value to an entry (i, j) of the matrix.
- T_ **operator()** (size_t **i**, size_t **j**) const
Operator () (Constant version).
- T_ & **operator()** (size_t **i**, size_t **j**)
Operator () (Non constant version).
- **BMatrix**< T_ > & **operator=** (const **BMatrix**< T_ > &**m**)
Operator =.
- **BMatrix**< T_ > & **operator=** (const T_ &**x**)
Operator =.
- **BMatrix**< T_ > & **operator*=** (const T_ &**x**)
*Operator *=.*
- **BMatrix**< T_ > & **operator+=** (const T_ &**x**)
Operator +=.
- int **Factor** ()
Factorize matrix.
- int **Solve** (**Vect**< T_ > &**b**)
Solve system with current matrix (forward and back substitution).
- T_ * **get** () const
Return C-Array.
- T_ **get** (size_t **i**, size_t **j**) const

- Return entry (i, j) of matrix.*

 - size_t `getNbRows` () const

Return number of rows.
- size_t `getNbColumns` () const

Return number of columns.
- void `setPenal` (real_t p)

Set Penalty Parameter (For boundary condition prescription).
- void `setDiagonal` ()

Set the matrix as diagonal.
- void `setDiagonal` (const class `Mesh` &mesh)

Initialize matrix storage in the case where only diagonal terms are stored.
- T_ `getDiag` (size_t k) const

Return k-th diagonal entry of matrix.
- size_t `size` () const

Return matrix dimension (Number of rows and columns).
- void `Assembly` (const class `Element` *el, T_ *a)

Assembly of element matrix into global matrix.
- void `Assembly` (const class `Element` *el, const `DMatrix`< T_ > &a)

Assembly of element matrix into global matrix.
- void `Assembly` (const class `Side` *sd, T_ *a)

Assembly of side matrix into global matrix.
- void `Assembly` (const class `Side` *sd, const `DMatrix`< T_ > &a)

Assembly of side matrix into global matrix.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition.
- void `Prescribe` (int dof, int code, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- void `Prescribe` (`Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, int flag=0)

Impose by a penalty method a homogeneous (=0) essential boundary condition.
- void `Prescribe` (`Vect`< T_ > &b, int flag=0)

Impose by a penalty method a homogeneous (=0) essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- int `Solve` (const `Vect`< T_ > &b, `Vect`< T_ > &x)

- *Solve system with factorized matrix (forward and back substitution).*
• int **FactorAndSolve** (**Vect**< T_ > &b)
- *Factorize matrix and solve the linear system.*
• int **FactorAndSolve** (const **Vect**< T_ > &b, **Vect**< T_ > &x)
- *Factorize matrix and solve the linear system.*
• unsigned long **getLength** () const
- *Return number of stored terms in matrix.*
• int **isDiagonal** () const
- *Say if matrix is diagonal or not.*
• int **isFactorized** () const
- *Say if matrix is factorized or not.*
• virtual size_t **getColInd** (size_t i) const
- *Return Column index for column i (See the description for class **SpMatrix**).*
• virtual size_t **getRowPtr** (size_t i) const
- *Return Row pointer for row i (See the description for class **SpMatrix**).*
• T_ **operator()** (size_t i) const
- *Operator () with one argument (Constant version).*
• T_ & **operator()** (size_t i)
- *Operator () with one argument (Non Constant version).*
• T_ & **operator[]** (size_t k)
- *Operator [] (Non constant version).*
• T_ **operator[]** (size_t k) const
- *Operator [] (Constant version).*
• **Matrix** & **operator+=** (const **Matrix**< T_ > &m)
- *Operator +=.*
• **Matrix** & **operator-=** (const **Matrix**< T_ > &m)
- *Operator -=.*
• **Matrix** & **operator-=** (const T_ &x)
- *Operator -=.*
• void **setPrintView** (size_t rmin, size_t rmax, size_t cmin, size_t cmax)
- *Set a window for matrix printing.*
• void **getPrintView** (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
- *Return window data for matrix printing.*

7.5.1 Detailed Description

template<class T_>class OFELI::BMatrix< T_ >

To handle band matrices.

This class enables storing and manipulating band matrices. The matrix can have different numbers of lower and upper co-diagonals

Template Parameters

< T_ >	Data type (double, float, complex<double>, ...)
---------------------	---

7.5.2 Constructor & Destructor Documentation

BMatrix ()

Default constructor.

Initialize a zero dimension band matrix

BMatrix (size_t size, int ld, int ud)

Constructor that for a band matrix with given size and bandwidth.

Assign 0 to all matrix entries.

Parameters

in	<i>size</i>	Number of rows and columns
in	<i>ld</i>	Number of lower co-diagonals (must be > 0)
in	<i>ud</i>	Number of upper co-diagonals (must be > 0)

7.5.3 Member Function Documentation**void setSize (size_t size, int ld, int ud)**

Set size (number of rows) and storage of matrix.

Parameters

in	<i>size</i>	Number of rows and columns
in	<i>ld</i>	Number of lower co-diagonals (must be > 0)
in	<i>ud</i>	Number of upper co-diagonals (must be > 0)

void Axy (T_ a, const BMatrix< T_ > & x)

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>x</i>	Matrix by which a is multiplied. The result is added to current instance

void Axy (T_ a, const Matrix< T_ > * x) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>x</i>	Matrix by which a is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

T_ operator() (size_t i, size_t j) const [virtual]

Operator () (Constant version).

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).

T_ & operator() (size_t i, size_t j) [virtual]

Operator () (Non constant version).

Parameters

<code>in</code>	<code>i</code>	Row index
<code>in</code>	<code>j</code>	Column index

Implements [Matrix< T_ >](#).

BMatrix< T_ > & operator= (const BMatrix< T_ > & m)

Operator =.

Copy matrix `m` to current matrix instance.

BMatrix< T_ > & operator= (const T_ & x)

Operator =.

Assign constant value `x` to all matrix entries.

BMatrix< T_ > & operator*= (const T_ & x)

Operator *.

Premultiply matrix entries by constant value `x`

BMatrix< T_ > & operator+= (const T_ & x)

Operator +.

Add constant `x` to matrix entries.

int Factor () [virtual]

Factorize matrix.

Perform LU factorization of matrix. Entries of matrix are modified after use of this function

Returns

0 if factorization is successful, `i` means that the `i`-th pivot was null.

Implements [Matrix< T_ >](#).

int Solve (Vect< T_ > & b) [virtual]

Solve system with current matrix (forward and back substitution).

Parameters

<code>in</code>	<code>b</code>	Vect instance that contains right-hand side on input and solution on output.
-----------------	----------------	--

Returns

- 0 if solution was normally performed
- `n` if the `n`-th pivot is null.

Warning

Matrix is modified after this function.

Implements [Matrix< T_ >](#).

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return k-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homogeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<code>in</code>	<code>mesh</code>	Mesh instance from which information is extracted.
<code>in,out</code>	<code>b</code>	Vect instance that contains right-hand side.
<code>in</code>	<code>flag</code>	Parameter to determine whether only the right-hand side is to be modified (<code>dof>0</code>) or both matrix and right-hand side (<code>dof=0</code> , default value).

void Prescribe ([Vect](#)< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homogeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function [setPenal](#)(..).

Parameters

<code>in,out</code>	<code>b</code>	Vect instance that contains right-hand side.
<code>in</code>	<code>flag</code>	Parameter to determine whether only the right-hand side is to be modified (<code>dof>0</code>) or both matrix and right-hand side (<code>dof=0</code> , default value).

void Prescribe (size_t dof, const class [Mesh](#) & mesh, [Vect](#)< T_ > & b, const [Vect](#)< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function [setPenal](#).

Parameters

<code>in</code>	<code>dof</code>	Label of the concerned degree of freedom (DOF).
<code>in</code>	<code>mesh</code>	Mesh instance from which information is extracted.
<code>in,out</code>	<code>b</code>	Vect instance that contains right-hand side.
<code>in</code>	<code>u</code>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<code>in</code>	<code>flag</code>	Parameter to determine whether only the right-hand side is to be modified (<code>dof>0</code>) or both matrix and right-hand side (<code>dof=0</code> , default value).

void Prescribe (size_t dof, [Vect](#)< T_ > & b, const [Vect](#)< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function [setPenal](#).

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

int Solve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Solve system with factorized matrix (forward and back substitution).

Parameters

<i>in</i>	<i>b</i>	Vect instance that contains right-hand side
<i>out</i>	<i>x</i>	Vect instance that contains solution

Returns

- - 0 if solution was normally performed
 - *n* if the *n*-th pivot is null
- Solution is performed only if factorization has previously been invoked.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
---------------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.

Matrix< T_ > & operator-= (const Matrix< T_ > & m) [inherited]

Operator -=.

Subtract matrix *m* from current matrix instance.

Matrix< T_ > & operator-= (const T_ & x) [inherited]

Operator -=.

Subtract constant value x from all matrix entries.

void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax) [inherited]

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void getView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const [inherited]

Return window data for matrix printing.

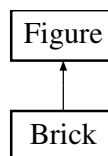
Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.6 Brick Class Reference

To store and treat a brick (parallelepiped) figure.

Inheritance diagram for Brick:



Public Member Functions

- **Brick ()**
Default constructor.
- **Brick (const Point< double > &bbm, const Point< double > &bbM, int code=1)**
Constructor.
- **void setBoundingBox (const Point< double > &bbm, const Point< double > &bbM)**
Assign bounding box of the brick.
- **Point< double > getBoundingBox1 () const**
Return first point of bounding box (xmin,ymin,zmin)
- **Point< double > getBoundingBox2 () const**
Return second point of bounding box (xmax,ymax,zmax)
- **double getSignedDistance (const Point< double > &p) const**
Return signed distance of a given point from the current brick.

- **Brick** & **operator+=** (**Point**< double > a)
Operator +=.
- **Brick** & **operator+=** (double a)
*Operator *+=.*
- void **setCode** (int code)
Choose a code for the domain defined by the figure.
- void **getSignedDistance** (const **Grid** &g, **Vect**< double > &d) const
Calculate signed distance to current figure with respect to grid points.
- double **dLine** (const **Point**< double > &p, const **Point**< double > &a, const **Point**< double > &b) const
Compute signed distance from a line.

7.6.1 Detailed Description

To store and treat a brick (parallelepiped) figure.

7.6.2 Constructor & Destructor Documentation

Brick (const **Point**< double > &bbm, const **Point**< double > &bbM, int code = 1)

Constructor.
Parameters

in	<i>bbm</i>	first point (xmin,ymin,zmin)
in	<i>bbM</i>	second point (xmax,ymax,zmax)
in	<i>code</i>	Code to assign to rectangle

7.6.3 Member Function Documentation

void setBoundingBox (const **Point**< double > &bbm, const **Point**< double > &bbM)

Assign bounding box of the brick.
Parameters

in	<i>bbm</i>	first point (xmin,ymin,zmin)
in	<i>bbM</i>	second point (xmax,ymax,zmax)

double getSignedDistance (const **Point**< double > &p) const [virtual]

Return signed distance of a given point from the current brick.

The computed distance is negative if p lies in the brick, negative if it is outside, and 0 on its boundary

Parameters

in	<i>p</i>	Point <double> instance
----	----------	--------------------------------

Reimplemented from [Figure](#).

Brick& operator+= (**Point**< double > a)

Operator +=.

Translate brick by a vector a

Brick& operator+= (double a)

Operator *+=.

Scale brick by a factor a

void getSignedDistance (const Grid & g, Vect< double > & d) const [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector d doesn't need to be sized before invoking this function

double dLine (const Point< double > & p, const Point< double > & a, const Point< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	<i>p</i>	Point for which distance is computed
in	<i>a</i>	First vertex of line
in	<i>b</i>	Second vertex of line

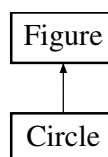
Returns

Signed distance

7.7 Circle Class Reference

To store and treat a circular figure.

Inheritance diagram for Circle:



Public Member Functions

- [Circle](#) ()
Default constructor.
- [Circle](#) (const [Point](#)< double > &c, double r, int code=1)
Constructor.
- void [setRadius](#) (double r)
Assign radius of circle.
- double [getRadius](#) () const
Return radius of circle.

- void `setCenter` (const `Point< double >` &c)
Assign coordinates of center of circle.
- `Point< double >` `getCenter` () const
Return coordinates of center of circle.
- double `getSignedDistance` (const `Point< double >` &p) const
Return signed distance of a given point from the current circle.
- `Circle &` `operator+=` (`Point< double >` a)
Operator +=.
- `Circle &` `operator+=` (double a)
*Operator *=.*
- void `setCode` (int code)
Choose a code for the domain defined by the figure.
- void `getSignedDistance` (const `Grid &`g, `Vect< double >` &d) const
Calculate signed distance to current figure with respect to grid points.
- double `dLine` (const `Point< double >` &p, const `Point< double >` &a, const `Point< double >` &b) const
Compute signed distance from a line.

7.7.1 Detailed Description

To store and treat a circular figure.

7.7.2 Constructor & Destructor Documentation

Circle (const `Point< double >` & c, double r, int code = 1)

Constructor.
Parameters

in	<i>c</i>	Coordinates of center of circle
in	<i>r</i>	Radius
in	<i>code</i>	Code to assign to the generated domain [Default: 1]

7.7.3 Member Function Documentation

double `getSignedDistance` (const `Point< double >` & p) const [virtual]

Return signed distance of a given point from the current circle.

The computed distance is negative if p lies in the disk, positive if it is outside, and 0 on the circle
Parameters

in	<i>p</i>	<code>Point<double></code> instance
----	----------	---

Reimplemented from [Figure](#).

`Circle&` `operator+=` (`Point< double >` a)

Operator +=.

Translate circle by a vector a

`Circle&` `operator+=` (double a)

Operator *=.

Scale circle by a factor a

void getSignedDistance (const Grid & g, Vect< double > & d) const [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector *d* doesn't need to be sized before invoking this function

double dLine (const Point< double > & p, const Point< double > & a, const Point< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	<i>p</i>	Point for which distance is computed
in	<i>a</i>	First vertex of line
in	<i>b</i>	Second vertex of line

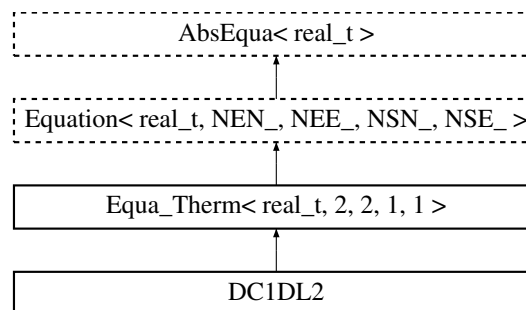
Returns

Signed distance

7.8 DC1DL2 Class Reference

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.

Inheritance diagram for DC1DL2:



Public Member Functions

- [DC1DL2](#) ()
Default Constructor.
- [DC1DL2](#) (const [Element](#) *el)
Constructor for an element.
- [DC1DL2](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for an element (transient case).
- [DC1DL2](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for an element (transient case) with specification of time integration scheme.
- [~DC1DL2](#) ()
Destructor.

- void `build` ()
Build the linear system without solving.
- void `LCapacityToLHS` (`real_t` coef=1)
Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef
- void `LCapacityToRHS` (`real_t` coef=1)
Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef
- void `LCapacity` (`real_t` coef)
Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient coef
- void `CapacityToLHS` (`real_t` coef=1)
Add Consistent capacity matrix to left-hand side after multiplying it by coefficient coef.
- void `CapacityToRHS` (`real_t` coef=1)
Add Consistent capacity contribution to right-hand side after multiplying it by coefficient coef
- void `Capacity` (`real_t` coef=1)
Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient coef
- void `Diffusion` (`real_t` coef=1)
Add diffusion matrix to left hand side after multiplying it by coefficient coef
- void `DiffusionToRHS` (`real_t` coef=1)
Add diffusion contribution to right hand side after multiplying it by coefficient coef
- void `Convection` (const `real_t` &v, `real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `Convection` (const `Vect`< `real_t` > &v, `real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `Convection` (`real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `ConvectionToRHS` (const `real_t` &v, `real_t` coef=1)
Add convection contribution to right-hand side after multiplying it by coefficient coef
- void `ConvectionToRHS` (`real_t` coef=1)
Add convection contribution to right-hand side after multiplying it by coefficient coef
- void `BodyRHS` (`UserData`< `real_t` > &ud, `real_t` coef=1)
Add body right-hand side term to right hand side after multiplying it by coefficient coef
- void `BodyRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
Add body right-hand side term to right hand side.
- void `BoundaryRHS` (`UserData`< `real_t` > &ud, `real_t` coef=1)
Add boundary right-hand side term to right hand side after multiplying it by coefficient coef
- void `BoundaryRHS` (`real_t` flux)
Add boundary right-hand side flux to right hand side.
- void `BoundaryRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
Add boundary right-hand side term to right hand side after multiplying it by coefficient coef
- `real_t` `Flux` () const
Return (constant) heat flux in element.
- void `setInput` (`EqDataType` opt, `Vect`< `real_t` > &u)
Set equation input data.
- virtual void `setStab` ()
Set stabilized formulation.
- void `setLumpedCapacity` ()

- Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.*

 - void `setCapacity ()`
- Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.*

 - void `setDiffusion ()`
- Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.*

 - void `setConvection ()`
- Add convection contribution to left and/or right-hand side taking into account time integration scheme.*

 - int `runTransient ()`
- Run one time step.*

 - int `run ()`
- Run the equation.*

 - void `setRhoCp (const real_t &rhocp)`
- Set product of Density by Specific heat (constants)*

 - void `setConductivity (const real_t &diff)`
- Set (constant) thermal conductivity.*

 - void `RhoCp (const string &exp)`
- Set product of Density by Specific heat given by an algebraic expression.*

 - void `Conduc (const string &exp)`
- Set thermal conductivity given by an algebraic expression.*

 - void `buildEigen (SkSMatrix< real_t > &K, SkSMatrix< real_t > &M)`
- Build global stiffness and mass matrices for the eigen system.*

 - void `buildEigen (SkSMatrix< real_t > &K, Vect< real_t > &M)`
- Build global diffusion and capacity matrices for the eigen system.*

 - void `updateBC (const Element &el, const Vect< real_t > &bc)`
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `updateBC (const Vect< real_t > &bc)`
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `DiagBC (int dof_type=NODE_DOF, int dof=0)`
- Update element matrix to impose bc by diagonalization technique.*

 - void `LocalNodeVector (Vect< real_t > &b)`
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEE_ > &be)`
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be, int dof)`
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVectorSingleDOF (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be)`
- Localize Element Vector from a Vect instance.*

 - void `ElementSideVector (const Vect< real_t > &b, LocalVect< real_t, NSE_ > &be)`
- Localize Element Vector from a Vect instance.*

 - void `ElementVector (const Vect< real_t > &b, int dof_type=NODE_FIELD, int flag=0)`
- Localize Element Vector.*

 - void `SideVector (const Vect< real_t > &b)`
- Localize Side Vector.*

 - void `ElementNodeCoordinates ()`

- Localize coordinates of element nodes.*
- void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*
- void `ElementAssembly` (`Matrix`< `real_t` > *A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`TrMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`Vect`< `real_t` > &v)
- Assemble element vector into global one.*
- void `DGElementAssembly` (`Matrix`< `real_t` > *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`TrMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `SideAssembly` (`Matrix`< `real_t` > *A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`Vect`< `real_t` > &v)
- Assemble side (edge or face) vector into global one.*
- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
- Assemble product of element matrix by element vector into global vector.*
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
- Assemble product of side matrix by side vector into global vector.*
- size_t `getNbNodes` () const
- Return number of element nodes.*
- size_t `getNbEq` () const
- Return number of element equations.*
- `real_t` * `A` ()
- Return element matrix as a C-array.*
- `real_t` * `sA` ()

- Return side matrix as a C-array.*

 - `real_t * b ()`

Return element right-hand side as a C-array.

 - `real_t * sb ()`

Return side right-hand side as a C-array.

 - `real_t * Prev ()`

Return element matrix as a C-array.

 - `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`

Return element matrix as a LocalMatrix instance.

 - `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`

Return side matrix as a LocalMatrix instance.

 - `LocalVect< real_t, NEE_ > & Eb ()`

Return element right-hand side as a LocalVect instance.

 - `LocalVect< real_t, NEE_ > & Ep ()`

Return element matrix as a C-array.

 - `void setInitialSolution (const Vect< real_t > &u)`

Set initial solution (previous time step)

 - `real_t setMaterialProperty (const string &exp, const string &prop)`

Define a material property by an algebraic expression.

 - `void setMesh (class Mesh &m)`

Define mesh and renumber DOFs after removing imposed ones.

 - `Mesh & getMesh () const`

Return reference to Mesh instance.

 - `LinearSolver< real_t > & getLinearSolver ()`

Return reference to linear solver instance.

 - `void setSolver (int ls, int pc=IDENT_PREC)`

Choose solver for the linear system.

 - `int solveEigenProblem (int nb_eigv, bool g=false)`

Compute eigenvalues and eigenvectors.

 - `real_t getEigenValue (int n) const`

Return the n-th eigenvalue.

 - `void getEigenVector (int n, Vect< real_t > &v) const`

Store the eigenvector corresponding to a given eigenvalue.

 - `class Eigen & getEigenSolver ()`

Return reference to eigenproblem solver.

Protected Member Functions

- `void setMaterial ()`
- Set material properties.*
- `void Init (const Element *el)`
- Set element arrays to zero.*
- `void Init (const Side *sd)`
- Set side arrays to zero.*

7.8.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 1-D using 2-Node elements.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.8.2 Constructor & Destructor Documentation

DC1DL2 ()

Default Constructor.

Constructs an empty equation.

DC1DL2 (const Element * el, const Vect< real_t > & u, real_t time = 0.)

Constructor for an element (transient case).

Parameters

<i>el</i>	[in]	Pointer to element
<i>u</i>	[in]	Vect instance that contains solution at previous time step
<i>time</i>	[in]	Current time value (Default value is 0)

DC1DL2 (const Element * el, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value (Default value is 0).
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme: <ul style="list-style-type: none"> • FORWARD_EULER for Forward Euler scheme • BACKWARD_EULER for Backward Euler scheme • CRANK_NICOLSON for Crank-Nicolson Euler scheme

7.8.3 Member Function Documentation

void LCapacityToLHS (real_t coef = 1) [virtual]

Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
----	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void LCapacityToRHS (real_t coef = 1) [virtual]

Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void LCapacity (real_t coef)

Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

void CapacityToLHS (real_t coef = 1) [virtual]

Add Consistent capacity matrix to left-hand side after multiplying it by coefficient *coef*.

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void CapacityToRHS (real_t coef = 1) [virtual]

Add Consistent capacity contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void Capacity (real_t coef = 1)

Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

void Diffusion (real_t coef = 1) [virtual]

Add diffusion matrix to left hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void DiffusionToRHS (real_t coef = 1) [virtual]

Add diffusion contribution to right hand side after multiplying it by coefficient *coef*

To be used for explicit diffusion term

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void Convection (const real_t & v, real_t coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>v</i>	Constant velocity vector
in	<i>coef</i>	Coefficient to multiply by added term [default: 1]

void Convection (const Vect< real_t > & v, real_t coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field is given by a vector *v*

Parameters

in	<i>v</i>	Velocity vector
in	<i>coef</i>	Coefficient to multiply by added term [default: 1]

void Convection (real_t coef = 1) [virtual]

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
----	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void ConvectionToRHS (const real_t & v, real_t coef = 1)

Add convection contribution to right-hand side after multiplying it by coefficient *coef*

To be used for explicit convection term.

Parameters

in	<i>v</i>	Velocity vector
in	<i>coef</i>	Coefficient to multiply by added term [default: 1]

void ConvectionToRHS (real_t coef = 1) [virtual]

Add convection contribution to right-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [default: 1]
----	-------------	--

Reimplemented from [Equa_Therm< real_t, 2, 2, 1, 1 >](#).

void BodyRHS (UserData< real_t > & ud, real_t coef = 1)

Add body right-hand side term to right hand side after multiplying it by coefficient *coef*

Parameters

in	<i>ud</i>	Instance of UserData or of a derived class. Contains a member function that provides body source.
in	<i>coef</i>	Coefficient to multiply by added term [default: 1]

void BodyRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>b</i>	Vector containing source at element nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BoundaryRHS (UserData< real_t > & ud, real_t coef = 1)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Parameters

in	<i>ud</i>	Instance of UserData or of an inherited class. Contains a member function that provides body source.
in	<i>coef</i>	Coefficient to multiply by added term [default: 1]

void BoundaryRHS (real_t flux)

Add boundary right-hand side flux to right hand side.

Parameters

in	<i>flux</i>	Vector containing source at side nodes.
-----------	-------------	---

void BoundaryRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Parameters

in	<i>b</i>	Vector containing source at side nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void setInput (EqDataType opt, Vect< real_t > & u)

Set equation input data.

Parameters

in	<i>opt</i>	Parameter that selects data type for input. This parameter is to be chosen in the enumerated variable EqDataType <ul style="list-style-type: none"> • INITIAL_FIELD: Initial temperature • BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet) • SOURCE_DATA: Heat source • FLUX_DATA: Heat flux (Neumann boundary condition) • VELOCITY: Velocity vector (for the convection term)
in	<i>u</i>	Vector containing input data

virtual void setStab () [virtual], [inherited]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

int runTransient () [inherited]

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run () [inherited]

Run the equation.

If the analysis (see function setAnalysis) is *STEADY_STATE*, then the function solves the stationary equation.

If the analysis is *TRANSIENT*, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	K	Diffusion matrix
in	M	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<i>in</i>	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [Default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides
<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <i>dof</i> is handled in the system

void LocalNodeVector (Vect< real.t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <i>ePrev</i> . This member function is to be used if a constructor with <i>Element</i> was invoked.
-----------	----------	---

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real_t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer `_theElement`

void SideVector (const Vect< real_t > & b) [inherited]

Localize Side Vector.

Parameters

<code>in</code>	<code>b</code>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------------	----------------	---

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

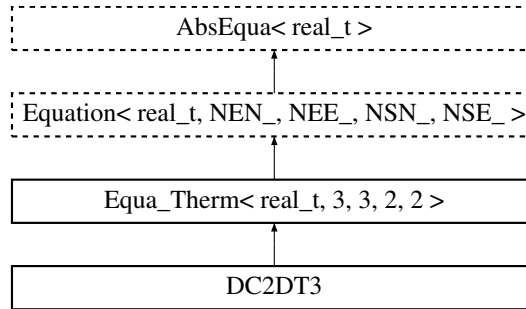
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.9 DC2DT3 Class Reference

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Inheritance diagram for DC2DT3:



Public Member Functions

- [DC2DT3](#) ()
Default Constructor. Constructs an empty equation.
- [DC2DT3](#) ([Mesh](#) &ms)
Constructor.
- [DC2DT3](#) ([Mesh](#) &ms, [Vect](#)< [real_t](#) > &u)
Constructor.
- [DC2DT3](#) (const [Element](#) *el)
Constructor for an element.
- [DC2DT3](#) (const [Side](#) *sd)
Constructor for a boundary side.
- [DC2DT3](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for an element (transient case).
- [DC2DT3](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for an element (transient case) with specification of time integration scheme.
- [DC2DT3](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for a boundary side (transient case).
- [DC2DT3](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for a side (transient case) with specification of time integration scheme.
- [~DC2DT3](#) ()
Destructor.
- void [LCapacityToLHS](#) ([real_t](#) coef=1)
Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef
- void [LCapacityToRHS](#) ([real_t](#) coef=1)
Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef
- void [LCapacity](#) ([real_t](#) coef)

- Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient `coef`*

 - void `CapacityToLHS` (`real_t` `coef`=1)
- Add Consistent capacity matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `CapacityToRHS` (`real_t` `coef`=1)
- Add Consistent capacity contribution to right-hand side after multiplying it by coefficient `coef`*

 - void `Capacity` (`real_t` `coef`=1)
- Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient `coef`*

 - void `Diffusion` (`real_t` `coef`=1)
- Add diffusion matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `Diffusion` (const `LocalMatrix`< `real_t`, 2, 2 > &diff, `real_t` `coef`=1)
- Add diffusion matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `DiffusionToRHS` (`real_t` `coef`=1)
- Add diffusion contribution to right-hand side after multiplying it by coefficient `coef` To be used for explicit diffusion term.*

 - void `Convection` (const `Point`< `real_t` > &v, `real_t` `coef`=1)
- Add convection matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `Convection` (const `Vect`< `real_t` > &v, `real_t` `coef`=1)
- Add convection matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `Convection` (`real_t` `coef`=1)
- Add convection matrix to left-hand side after multiplying it by coefficient `coef`*

 - void `ConvectionToRHS` (const `Point`< `real_t` > &v, `real_t` `coef`=1)
- Add convection contribution to right-hand side after multiplying it by coefficient `coef`*

 - void `ConvectionToRHS` (`real_t` `coef`=1)
- Add convection contribution to right-hand side after multiplying it by coefficient `coef`*

 - void `LinearExchange` (`real_t` `coef`, `real_t` T)
- Add an edge linear exchange term to left and right-hand sides.*

 - void `BodyRHS` (`UserData`< `real_t` > &ud, `real_t` `coef`=1)
- Add body right-hand side term to right hand side after multiplying it by coefficient `coef`*

 - void `BodyRHS` (const `Vect`< `real_t` > &bf, int opt=`GLOBAL_ARRAY`)
- Add body right-hand side term to right hand side.*

 - void `BodyRHS` (`real_t` bf)
- Add body right-hand side term to right hand side.*

 - void `BoundaryRHS` (`UserData`< `real_t` > &ud, `real_t` `coef`=1)
- Add boundary right-hand side term to right hand side after multiplying it by coefficient `coef`*

 - void `BoundaryRHS` (`real_t` flux)
- Add boundary right-hand side flux to right hand side.*

 - void `BoundaryRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
- Add boundary right-hand side term to right hand side after multiplying it by coefficient `coef`*

 - void `Periodic` (`real_t` `coef`=1.e20)
- Add contribution of periodic boundary condition (by a penalty technique).*

 - `Point`< `real_t` > & `Flux` () const
- Return (constant) heat flux in element.*

 - `Point`< `real_t` > & `Grad` (const `LocalVect`< `real_t`, 3 > &u) const
- Return gradient of a vector in element.*

 - `Point`< `real_t` > & `Grad` (const `Vect`< `real_t` > &u) const
- Return gradient of a vector in element.*

- void `setInput` (`EqDataType` opt, `Vect< real_t >` &u)
Set equation input data.
- void `JouleHeating` (const `Vect< real_t >` &sigma, const `Vect< real_t >` &psi)
Set Joule heating term as source.
- void `build` ()
Build the linear system without solving.
- int `runSteadyState` ()
Solve the equations in the steady state case.
- virtual void `setStab` ()
Set stabilized formulation.
- void `setLumpedCapacity` ()
Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setCapacity` ()
Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setDiffusion` ()
Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.
- void `setConvection` ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- int `runTransient` ()
Run one time step.
- int `run` ()
Run the equation.
- void `setRhoCp` (const `real_t` &rhocp)
Set product of Density by Specific heat (constants)
- void `setConductivity` (const `real_t` &diff)
Set (constant) thermal conductivity.
- void `RhoCp` (const string &exp)
Set product of Density by Specific heat given by an algebraic expression.
- void `Conduc` (const string &exp)
Set thermal conductivity given by an algebraic expression.
- void `buildEigen` (`SkSMatrix< real_t >` &K, `SkSMatrix< real_t >` &M)
Build global stiffness and mass matrices for the eigen system.
- void `buildEigen` (`SkSMatrix< real_t >` &K, `Vect< real_t >` &M)
Build global diffusion and capacity matrices for the eigen system.
- void `updateBC` (const `Element` &el, const `Vect< real_t >` &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect< real_t >` &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC` (int dof.type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector` (`Vect< real_t >` &b)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEE_ >` &be)
Localize Element Vector from a Vect instance.

- void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NSE_ >` &be)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect< real_t >` &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect< real_t >` &b)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix< real_t >` *A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix< real_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< real_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< real_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< real_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< real_t >` &v)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< real_t >` *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix< real_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< real_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< real_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix< real_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix< real_t >` *A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix< real_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< real_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix< real_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect< real_t >` &v)
Assemble side (edge or face) vector into global one.

- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of side matrix by side vector into global vector.
- `size_t` `getNbNodes` () const
Return number of element nodes.
- `size_t` `getNbEq` () const
Return number of element equations.
- `real_t` * `A` ()
Return element matrix as a C-array.
- `real_t` * `sA` ()
Return side matrix as a C-array.
- `real_t` * `b` ()
Return element right-hand side as a C-array.
- `real_t` * `sb` ()
Return side right-hand side as a C-array.
- `real_t` * `Prev` ()
Return element matrix as a C-array.
- `LocalMatrix`< `real_t`, `NEE_`, `NEE_` > & `EA` ()
Return element matrix as a LocalMatrix instance.
- `LocalMatrix`< `real_t`, `NSE_`, `NSE_` > & `SA` ()
Return side matrix as a LocalMatrix instance.
- `LocalVect`< `real_t`, `NEE_` > & `Eb` ()
Return element right-hand side as a LocalVect instance.
- `LocalVect`< `real_t`, `NEE_` > & `Ep` ()
Return element matrix as a C-array.
- void `setInitialSolution` (const `Vect`< `real_t` > &u)
Set initial solution (previous time step)
- `real_t` `setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.
- `LinearSolver`< `real_t` > & `getLinearSolver` ()
Return reference to linear solver instance.
- void `setSolver` (int ls, int pc=`IDENT_PREC`)
Choose solver for the linear system.
- int `solveEigenProblem` (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- `real_t` `getEigenValue` (int n) const
Return the n-th eigenvalue.
- void `getEigenvector` (int n, `Vect`< `real_t` > &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class `Eigen` & `getEigenSolver` ()
Return reference to eigenproblem solver.

Protected Member Functions

- void `setMaterial` ()
Set material properties.
- void `Init` (const `Element` *el)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.9.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.9.2 Constructor & Destructor Documentation

DC2DT3 (const `Element` * el, const `Vect`< `real_t` > & u, `real_t` time = 0.)

Constructor for an element (transient case).

Parameters

in	<i>el</i>	Pointer to element
in	<i>u</i>	<code>Vect</code> instance that contains solution at previous time step
in	<i>time</i>	Current time value [Default: 0]

DC2DT3 (const `Element` * el, const `Vect`< `real_t` > & u, `real_t` time, `real_t` deltat, int scheme)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	<code>Vect</code> instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].
in	<i>deltat</i>	Value of time step.
in	<i>scheme</i>	Time Integration Scheme: <ul style="list-style-type: none"> • FORWARD_EULER for Forward Euler scheme • BACKWARD_EULER for Backward Euler scheme • CRANK_NICOLSON for Crank-Nicolson Euler scheme

DC2DT3 (const `Side` * sd, const `Vect`< `real_t` > & u, `real_t` time = 0.)

Constructor for a boundary side (transient case).

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	<code>Vect</code> instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0]

DC2DT3 (**const Side * sd**, **const Vect< real_t > & u**, **real_t time**, **real_t deltat**, **int scheme**)

Constructor for a side (transient case) with specification of time integration scheme.

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].
in	<i>deltat</i>	Value of time step.
in	<i>scheme</i>	Time Integration Scheme: <ul style="list-style-type: none"> • FORWARD_EULER for Forward Euler scheme • BACKWARD_EULER for Backward Euler scheme • CRANK_NICOLSON for Crank-Nicolson Euler scheme

7.9.3 Member Function Documentation

void LCapacityToLHS (real_t coef = 1) [virtual]

Add lumped capacity matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
----	-------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void LCapacityToRHS (real_t coef = 1) [virtual]

Add lumped capacity contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
----	-------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void LCapacity (real_t coef)

Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
----	-------------	--

void CapacityToLHS (real_t coef = 1) [virtual]

Add Consistent capacity matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
----	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void CapacityToRHS (real_t coef = 1) [virtual]

Add Consistent capacity contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
----	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void Capacity (real_t coef = 1)

Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient coef

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

void Diffusion (*real_t* coef = 1) [virtual]

Add diffusion matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void Diffusion (const LocalMatrix< *real_t*, 2, 2 > & diff, *real_t* coef = 1)

Add diffusion matrix to left-hand side after multiplying it by coefficient *coef*

Case where the diffusivity matrix is given as an argument.

Parameters

<i>in</i>	<i>diff</i>	Diffusion matrix (class LocalMatrix).
<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1]

void DiffusionToRHS (*real_t* coef = 1) [virtual]

Add diffusion contribution to right-hand side after multiplying it by coefficient *coef* To be used for explicit diffusion term.

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void Convection (const Point< *real_t* > & v, *real_t* coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

<i>in</i>	<i>v</i>	Constant velocity vector
<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1]

void Convection (const Vect< *real_t* > & v, *real_t* coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field is given by a vector *v*

Parameters

<i>in</i>	<i>v</i>	Velocity vector
<i>in</i>	<i>coef</i>	Coefficient to multiply by added term (Default: 1)

void Convection (*real_t* coef = 1) [virtual]

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void ConvectionToRHS (const Point< real_t > & v, real_t coef = 1)

Add convection contribution to right-hand side after multiplying it by coefficient *coef*

To be used for explicit convection term.

Parameters

in	<i>v</i>	Velocity vector
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]

void ConvectionToRHS (real_t coef = 1) [virtual]

Add convection contribution to right-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void LinearExchange (real_t coef, real_t T)

Add an edge linear exchange term to left and right-hand sides.

Parameters

in	<i>coef</i>	Coefficient of exchange
in	<i>T</i>	External value for exchange

Remarks

This assumes a constant value of T

void BodyRHS (UserData< real_t > & ud, real_t coef = 1)

Add body right-hand side term to right hand side after multiplying it by coefficient *coef*

Parameters

in	<i>ud</i>	Instance of UserData or of a derived class. Contains a member function that provides body source.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]

void BodyRHS (const Vect< real_t > & bf, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>bf</i>	Vector containing source at element nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BodyRHS (real_t bf)

Add body right-hand side term to right hand side.

Case where the body right-hand side is piecewise constant.

Parameters

in	<i>bf</i>	Value of thermal source (Constant in element).
----	-----------	--

void BoundaryRHS (UserData< real_t > & ud, real_t coef = 1)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Parameters

in	<i>ud</i>	Instance of UserData or of an inherited class. Contains a member function that provides body source.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]

void BoundaryRHS (real_t flux)

Add boundary right-hand side flux to right hand side.

Parameters

in	<i>flux</i>	Vector containing source at side nodes.
----	-------------	---

void BoundaryRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Parameters

in	<i>b</i>	Vector containing source at side nodes
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void Periodic (real_t coef = 1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC_A on one side and PERIODIC_B on the opposite side.

Parameters

in	<i>coef</i>	Value of penalty parameter [Default: 1.e20]
----	-------------	---

Point<real_t>& Grad (const LocalVect< real_t, 3 > & u) const

Return gradient of a vector in element.

Parameters

in	<i>u</i>	Vector for which gradient is computed.
----	----------	--

Point<real_t>& Grad (const Vect< real_t > & u) const

Return gradient of a vector in element.

Parameters

in	<i>u</i>	Global vector for which gradient is computed. Vector <i>u</i> has as size the total number of nodes
-----------	----------	---

void setInput (EqDataType opt, Vect< real_t > & u)

Set equation input data.

Parameters

in	<i>opt</i>	Parameter to select type of input (enumerated values) <ul style="list-style-type: none"> • INITIAL_FIELD: Initial temperature • BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet) • SOURCE_DATA: Heat source • FLUX_DATA: Heat flux (Neumann boundary condition) • VELOCITY_FIELD: Velocity vector (for the convection term)
in	<i>u</i>	Vector containing input data

void JouleHeating (const Vect< real_t > & sigma, const Vect< real_t > & psi)

Set Joule heating term as source.

Parameters

in	<i>sigma</i>	Vect instance containing electric conductivity (elementwise)
in	<i>psi</i>	Vect instance containing electric potential (elementwise)

int runSteadyState ()

Solve the equations in the steady state case.

Returns

virtual void setStab () [virtual], [inherited]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

int runTransient () [inherited]

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run () [inherited]

Run the equation.

If the analysis (see function setAnalysis) is STEADY_STATE, then the function solves the stationary equation.

If the analysis is TRANSIENT, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	K	Diffusion matrix
in	M	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
----	------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector


```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMMatrix</code> instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

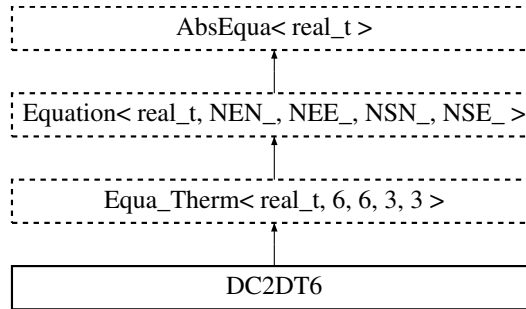
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.10 DC2DT6 Class Reference

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

Inheritance diagram for DC2DT6:



Public Member Functions

- [DC2DT6](#) ()
Default Constructor.
- [DC2DT6](#) (const [Element](#) *el)
Constructor for an element.
- [DC2DT6](#) (const [Side](#) *sd)
Constructor for a boundary side.
- [DC2DT6](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for an element (Transient case).
- [DC2DT6](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for an element (transient case) with specification of time integration scheme.
- [DC2DT6](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for a boundary side (transient case).
- [DC2DT6](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for a side (transient case) with specification of time integration scheme.
- [~DC2DT6](#) ()
Destructor.
- void [Diffusion](#) ([real_t](#) coef=1)
Add diffusion matrix to left hand side after multiplying it by coefficient coef
- void [Convection](#) ([real_t](#) coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void [Convection](#) ([Point](#)< [real_t](#) > &v, [real_t](#) coef=1)
Add convection matrix to left hand side after multiplying it by coefficient coef
- void [Convection](#) (const [Vect](#)< [real_t](#) > &v, [real_t](#) coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void [BodyRHS](#) (const [Vect](#)< [real_t](#) > &b, int opt=[GLOBAL_ARRAY](#))

- Add body right-hand side term to right hand side.*

 - void `BoundaryRHS` (const `Vect< real_t >` &b)
- Add boundary right-hand side term to right hand side after multiplying it by coefficient coef*

 - virtual void `setStab` ()
- Set stabilized formulation.*

 - virtual void `LCapacityToLHS` (`real_t` coef=1)
- Add lumped capacity contribution to left-hand side.*

 - virtual void `LCapacityToRHS` (`real_t` coef=1)
- Add lumped capacity contribution to right-hand side.*

 - virtual void `CapacityToLHS` (`real_t` coef=1)
- Add consistent capacity contribution to left-hand side.*

 - virtual void `CapacityToRHS` (`real_t` coef=1)
- Add consistent capacity contribution to right-hand side.*

 - void `setLumpedCapacity` ()
- Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.*

 - void `setCapacity` ()
- Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.*

 - virtual void `DiffusionToRHS` (`real_t` coef=1.)
- Add diffusion term to right-hand side.*

 - void `setDiffusion` ()
- Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.*

 - virtual void `ConvectionToRHS` (`real_t` coef=1.)
- Add convection term to right-hand side.*

 - void `setConvection` ()
- Add convection contribution to left and/or right-hand side taking into account time integration scheme.*

 - void `build` ()
- Build the linear system of equations.*

 - int `runTransient` ()
- Run one time step.*

 - int `run` ()
- Run the equation.*

 - void `setRhoCp` (const `real_t` &rhocp)
- Set product of Density by Specific heat (constants)*

 - void `setConductivity` (const `real_t` &diff)
- Set (constant) thermal conductivity.*

 - void `RhoCp` (const string &exp)
- Set product of Density by Specific heat given by an algebraic expression.*

 - void `Conduc` (const string &exp)
- Set thermal conductivity given by an algebraic expression.*

 - void `buildEigen` (`SkSMatrix< real_t >` &K, `SkSMatrix< real_t >` &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void `buildEigen` (`SkSMatrix< real_t >` &K, `Vect< real_t >` &M)
- Build global diffusion and capacity matrices for the eigen system.*

 - void `updateBC` (const `Element` &el, const `Vect< real_t >` &bc)

- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `updateBC` (const `Vect< real_t >` &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
- Update element matrix to impose bc by diagonalization technique.*

 - void `LocalNodeVector` (`Vect< real_t >` &b)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEE_ >` &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be, int dof)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVectorSingleDOF` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementSideVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NSE_ >` &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementVector` (const `Vect< real_t >` &b, int dof_type=NODE_FIELD, int flag=0)
- Localize Element Vector.*

 - void `SideVector` (const `Vect< real_t >` &b)
- Localize Side Vector.*

 - void `ElementNodeCoordinates` ()
- Localize coordinates of element nodes.*

 - void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*

 - void `ElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`Vect< real_t >` &v)
- Assemble element vector into global one.*

 - void `DGElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `SideAssembly` (`Matrix< real_t >` *A)

- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly (SkMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly (SkMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly (SpMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly (Vect< real_t > &v)`
- Assemble side (edge or face) vector into global one.*
 - void `AxbAssembly (const Element &el, const Vect< real_t > &x, Vect< real_t > &b)`
- Assemble product of element matrix by element vector into global vector.*
 - void `AxbAssembly (const Side &sd, const Vect< real_t > &x, Vect< real_t > &b)`
- Assemble product of side matrix by side vector into global vector.*
 - size_t `getNbNodes ()` const
- Return number of element nodes.*
 - size_t `getNbEq ()` const
- Return number of element equations.*
 - `real_t * A ()`
- Return element matrix as a C-array.*
 - `real_t * sA ()`
- Return side matrix as a C-array.*
 - `real_t * b ()`
- Return element right-hand side as a C-array.*
 - `real_t * sb ()`
- Return side right-hand side as a C-array.*
 - `real_t * Prev ()`
- Return element matrix as a C-array.*
 - `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
- Return element matrix as a LocalMatrix instance.*
 - `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
- Return side matrix as a LocalMatrix instance.*
 - `LocalVect< real_t, NEE_ > & Eb ()`
- Return element right-hand side as a LocalVect instance.*
 - `LocalVect< real_t, NEE_ > & Ep ()`
- Return element matrix as a C-array.*
 - void `setInitialSolution (const Vect< real_t > &u)`
- Set initial solution (previous time step)*
 - `real_t setMaterialProperty (const string &exp, const string &prop)`
- Define a material property by an algebraic expression.*
 - void `setMesh (class Mesh &m)`
- Define mesh and renumber DOFs after removing imposed ones.*
 - `Mesh & getMesh ()` const
- Return reference to Mesh instance.*
 - `LinearSolver< real_t > & getLinearSolver ()`
- Return reference to linear solver instance.*
 - void `setSolver (int ls, int pc=IDENT_PREC)`

- Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n -th eigenvalue.
- `void getEigenVector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void setMaterial ()`
Set material properties.
- `void Init (const Element *el)`
Set element arrays to zero.
- `void Init (const Side *sd)`
Set side arrays to zero.

7.10.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 6-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.10.2 Constructor & Destructor Documentation

DC2DT6 ()

Default Constructor.

Constructs an empty equation.

DC2DT6 (const Element * el)

Constructor for an element.

Parameters

<code>in</code>	<code>el</code>	Pointer to element.
-----------------	-----------------	---------------------

DC2DT6 (const Side * sd)

Constructor for a boundary side.

Parameters

<code>in</code>	<code>sd</code>	Pointer to side.
-----------------	-----------------	------------------

DC2DT6 (const Element * el, const Vect< real_t > & u, real_t time = 0.)

Constructor for an element (Transient case).

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].

DC2DT6 (const Element * el, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 1]
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme: <ul style="list-style-type: none"> • FORWARD_EULER: Forward Euler scheme • BACKWARD_EULER: Backward Euler scheme, • CRANK_NICOLSON: Crank-Nicolson Euler scheme.

DC2DT6 (const Side * sd, const Vect< real_t > & u, real_t time = 0.)

Constructor for a boundary side (transient case).

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].

DC2DT6 (const Side * sd, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for a side (transient case) with specification of time integration scheme.

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme: To be chosen among the enumerated values: <ul style="list-style-type: none"> • FORWARD_EULER: Forward Euler scheme • BACKWARD_EULER: Backward Euler scheme, • CRANK_NICOLSON: Crank-Nicolson Euler scheme.

7.10.3 Member Function Documentation

void Diffusion (`real_t` coef = 1) [virtual]

Add diffusion matrix to left hand side after multiplying it by coefficient coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 6, 6, 3, 3 >](#).

void Convection (real_t coef = 1) [virtual]

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 6, 6, 3, 3 >](#).

void Convection (Point< real_t > & v, real_t coef = 1)

Add convection matrix to left hand side after multiplying it by coefficient *coef*

Parameters

in	<i>v</i>	Constant velocity vector.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void Convection (const Vect< real_t > & v, real_t coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field is given by a vector *v*

Parameters

in	<i>v</i>	Velocity vector.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void BodyRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>b</i>	Local vector (of size 6) containing source at element nodes
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 6 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BoundaryRHS (const Vect< real_t > & b)

Add boundary right-hand side term to right hand side after multiplying it by coefficient *coef*

Parameters

in	<i>b</i>	Local vector (of size 3) containing source at side nodes.
-----------	----------	---

virtual void setStab () [virtual], [inherited]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

virtual void LCapacityToLHS (real_t coef = 1) [virtual], [inherited]

Add lumped capacity contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
-----------	-------------	--

virtual void LCapacityToRHS (real_t coef = 1) [virtual], [inherited]

Add lumped capacity contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

virtual void CapacityToLHS (real_t coef = 1) [virtual], [inherited]

Add consistent capacity contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
-----------	-------------	--

virtual void CapacityToRHS (real_t coef = 1) [virtual], [inherited]

Add consistent capacity contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

void build () [inherited]

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

int runTransient () [inherited]

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run () [inherited]

Run the equation.

If the analysis (see function setAnalysis) is *STEADY_STATE*, then the function solves the stationary equation.

If the analysis is *TRANSIENT*, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	K	Diffusion matrix
in	M	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
----	------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE_FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT_FIELD</code>, DOFs are supported by elements • <code>SIDE_FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
-----------	-----------------	--

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer `_theElement`

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
----	----------	--

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMatrix</code> instance
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>in</i>	<i>A</i>	Global matrix stored as an <code>SkMatrix</code> instance
-----------	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	sd	Reference to Side instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
-----------	-----------	---

in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
-----------	-----------	--

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real.t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

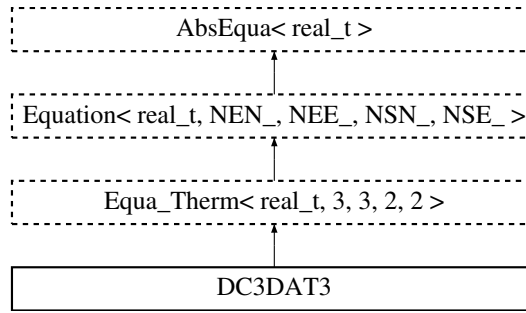
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.11 DC3DAT3 Class Reference

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

Inheritance diagram for DC3DAT3:



Public Member Functions

- **DC3DAT3** ()
Default Constructor.
- **DC3DAT3** (const **Element** *el)
Constructor for an element.
- **DC3DAT3** (const **Side** *sd)
Constructor for a boundary side.
- **DC3DAT3** (const **Element** *el, const **Vect**< **real_t** > &u, **real_t** time=0.)
Constructor for an element (transient case).
- **DC3DAT3** (const **Element** *el, const **Vect**< **real_t** > &u, **real_t** time, **real_t** deltat, int scheme)
Constructor for an element (transient case) with specification of time integration scheme.
- **DC3DAT3** (const **Side** *sd, const **Vect**< **real_t** > &u, **real_t** time=0.)
Constructor for a boundary side (transient case).
- **DC3DAT3** (const **Side** *sd, const **Vect**< **real_t** > &u, **real_t** time, **real_t** deltat, int scheme)
Constructor for a side (transient case) with specification of time integration scheme.
- **~DC3DAT3** ()
Destructor.
- void **LCapacityToLHS** (**real_t** coef=1)
Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef.
- void **LCapacityToRHS** (**real_t** coef=1)
Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef.
- void **LCapacity** (**real_t** coef=1)
Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient coef
- void **CapacityToLHS** (**real_t** coef=1)
Add Consistent capacity matrix to left-hand side after multiplying it by coefficientcoef
- void **CapacityToRHS** (**real_t** coef=1)
Add Consistent capacity contribution to right-hand side after multiplying it by coefficient coef.
- void **Capacity** (**real_t** coef=1)
Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient coef.
- void **Diffusion** (**real_t** coef=1)
Add diffusion matrix to left-hand side after multiplying it by coefficient coef
- void **Diffusion** (const **LocalMatrix**< **real_t**, 2, 2 > &diff, **real_t** coef=1)
Add diffusion matrix to left-hand side after multiplying it by coefficient coef
- void **DiffusionToRHS** (**real_t** coef=1)
Add diffusion contribution to right-hand side after multiplying it by coefficient coef

- void `BodyRHS` (`UserData`< `real_t` > &ud)
Add body right-hand side term to right-hand side after multiplying it by coefficient `coef`
- void `BodyRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
Add body right-hand side term to right hand side.
- void `BoundaryRHS` (`real_t` flux)
Add boundary right-hand side term to right hand side.
- `Point`< `real_t` > & `Grad` (const `Vect`< `real_t` > &u)
Return gradient of a vector in element.
- void `build` ()
Build the linear system without solving.
- virtual void `setStab` ()
Set stabilized formulation.
- void `setLumpedCapacity` ()
Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setCapacity` ()
Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setDiffusion` ()
Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.
- virtual void `Convection` (`real_t` coef=1.)
Add convection term to left-hand side.
- virtual void `ConvectionToRHS` (`real_t` coef=1.)
Add convection term to right-hand side.
- void `setConvection` ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- int `runTransient` ()
Run one time step.
- int `run` ()
Run the equation.
- void `setRhoCp` (const `real_t` &rhocp)
Set product of Density by Specific heat (constants)
- void `setConductivity` (const `real_t` &diff)
Set (constant) thermal conductivity.
- void `RhoCp` (const string &exp)
Set product of Density by Specific heat given by an algebraic expression.
- void `Conduc` (const string &exp)
Set thermal conductivity given by an algebraic expression.
- void `buildEigen` (`SkSMatrix`< `real_t` > &K, `SkSMatrix`< `real_t` > &M)
Build global stiffness and mass matrices for the eigen system.
- void `buildEigen` (`SkSMatrix`< `real_t` > &K, `Vect`< `real_t` > &M)
Build global diffusion and capacity matrices for the eigen system.
- void `updateBC` (const `Element` &el, const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.

- void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< [real_t](#) > &b)
Localize Element Vector from a Vect instance.
- void [ElementNodeVector](#) (const [Vect](#)< [real_t](#) > &b, [LocalVect](#)< [real_t](#), NEE_ > &be)
Localize Element Vector from a Vect instance.
- void [ElementNodeVector](#) (const [Vect](#)< [real_t](#) > &b, [LocalVect](#)< [real_t](#), NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void [ElementNodeVectorSingleDOF](#) (const [Vect](#)< [real_t](#) > &b, [LocalVect](#)< [real_t](#), NEN_ > &be)
Localize Element Vector from a Vect instance.
- void [ElementSideVector](#) (const [Vect](#)< [real_t](#) > &b, [LocalVect](#)< [real_t](#), NSE_ > &be)
Localize Element Vector from a Vect instance.
- void [ElementVector](#) (const [Vect](#)< [real_t](#) > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void [SideVector](#) (const [Vect](#)< [real_t](#) > &b)
Localize Side Vector.
- void [ElementNodeCoordinates](#) ()
Localize coordinates of element nodes.
- void [SideNodeCoordinates](#) ()
Localize coordinates of side nodes.
- void [ElementAssembly](#) ([Matrix](#)< [real_t](#) > *A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkSMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SpMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([TrMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([Vect](#)< [real_t](#) > &v)
Assemble element vector into global one.
- void [DGElementAssembly](#) ([Matrix](#)< [real_t](#) > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkSMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SpMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([TrMatrix](#)< [real_t](#) > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [SideAssembly](#) ([Matrix](#)< [real_t](#) > *A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SkSMatrix](#)< [real_t](#) > &A)
Assemble side (edge or face) matrix into global one.

- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect`< `real_t` > &v)
Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of side matrix by side vector into global vector.
- `size_t` `getNbNodes` () const
Return number of element nodes.
- `size_t` `getNbEq` () const
Return number of element equations.
- `real_t` * `A` ()
Return element matrix as a C-array.
- `real_t` * `sA` ()
Return side matrix as a C-array.
- `real_t` * `b` ()
Return element right-hand side as a C-array.
- `real_t` * `sb` ()
Return side right-hand side as a C-array.
- `real_t` * `Prev` ()
Return element matrix as a C-array.
- `LocalMatrix`< `real_t`, `NEE_`, `NEE_` > & `EA` ()
Return element matrix as a LocalMatrix instance.
- `LocalMatrix`< `real_t`, `NSE_`, `NSE_` > & `SA` ()
Return side matrix as a LocalMatrix instance.
- `LocalVect`< `real_t`, `NEE_` > & `Eb` ()
Return element right-hand side as a LocalVect instance.
- `LocalVect`< `real_t`, `NEE_` > & `Ep` ()
Return element matrix as a C-array.
- void `setInitialSolution` (const `Vect`< `real_t` > &u)
Set initial solution (previous time step)
- `real_t` `setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.
- `LinearSolver`< `real_t` > & `getLinearSolver` ()
Return reference to linear solver instance.
- void `setSolver` (int ls, int pc=`IDENT_PREC`)
Choose solver for the linear system.
- int `solveEigenProblem` (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.

- `real_t getEigenValue` (int n) const
Return the n-th eigenvalue.
- void `getEigenvector` (int n, `Vect< real_t >` &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class `Eigen` & `getEigenSolver` ()
Return reference to eigenproblem solver.

Protected Member Functions

- void `setMaterial` ()
Set material properties.
- void `Init` (const `Element` *el)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.11.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 3-D domains with axisymmetry using 3-Node triangles.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.11.2 Constructor & Destructor Documentation

DC3DAT3 ()

Default Constructor.

Constructs an empty equation.

DC3DAT3 (const `Element` * el)

Constructor for an element.

Parameters

<code>in</code>	<code>el</code>	Pointer to element.
-----------------	-----------------	---------------------

DC3DAT3 (const `Side` * sd)

Constructor for a boundary side.

Parameters

<code>in</code>	<code>sd</code>	Pointer to side.
-----------------	-----------------	------------------

DC3DAT3 (const `Element` * el, const `Vect< real_t >` & u, `real_t` time = 0.)

Constructor for an element (transient case).

Parameters

<code>in</code>	<code>el</code>	Pointer to element
<code>in</code>	<code>u</code>	<code>Vect</code> instance that contains solution at previous time step
<code>in</code>	<code>time</code>	Current time value [Default: 0]

DC3DAT3 (`const Element * el`, `const Vect< real_t > & u`, `real_t time`, `real_t deltat`, `int scheme`)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value.
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme (): <ul style="list-style-type: none"> • FORWARD_EULER for Forward Euler scheme, BACKWARD_EULER for Backward Euler scheme, • CRANK_NICOLSON for Crank-Nicolson Euler scheme.

DC3DAT3 (const Side * sd, const Vect< real_t > & u, real_t time = 0.)

Constructor for a boundary side (transient case).

Parameters

in	<i>sd</i>	Pointer to side
in	<i>u</i>	Vect instance that contains solution at previous time step
in	<i>time</i>	Current time value [Default: 0]

DC3DAT3 (const Side * sd, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for a side (transient case) with specification of time integration scheme.

Parameters

in	<i>sd</i>	Pointer to side
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value.
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme (enumerated values) : <ul style="list-style-type: none"> • FORWARD_EULER: Forward Euler scheme • BACKWARD_EULER: Backward Euler scheme • CRANK_NICOLSON: Crank-Nicolson Euler scheme

7.11.3 Member Function Documentation

void LCapacityToLHS (real_t coef = 1) [virtual]

Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef.

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
----	-------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void LCapacityToRHS (real_t coef = 1) [virtual]

Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef.

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void LCapacity (real_t coef = 1)

Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient `coef`

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

void CapacityToLHS (real_t coef = 1) [virtual]

Add Consistent capacity matrix to left-hand side after multiplying it by coefficient `coef`

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void CapacityToRHS (real_t coef = 1) [virtual]

Add Consistent capacity contribution to right-hand side after multiplying it by coefficient `coef`.

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void Capacity (real_t coef = 1)

Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient `coef`.

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

void Diffusion (real_t coef = 1) [virtual]

Add diffusion matrix to left-hand side after multiplying it by coefficient `coef`

Parameters

<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1].
-----------------	-------------------	---

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void Diffusion (const LocalMatrix< real_t, 2, 2 > & diff, real_t coef = 1)

Add diffusion matrix to left-hand side after multiplying it by coefficient `coef`

Case where the diffusivity matrix is given as an argument

Parameters

<code>in</code>	<code>diff</code>	Instance of class DMatrix containing diffusivity matrix
<code>in</code>	<code>coef</code>	Coefficient to multiply by added term [Default: 1]

void DiffusionToRHS (real_t coef = 1) [virtual]

Add diffusion contribution to right-hand side after multiplying it by coefficient `coef`
To be used for explicit diffusion term

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 3, 3, 2, 2 >](#).

void BodyRHS (UserData< real_t > & ud)

Add body right-hand side term to right-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>ud</i>	Instance of UserData or of an inherited class. Contains a member function that provides body source.
-----------	-----------	--

void BodyRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>b</i>	Local vector (of size 3) containing source at element nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BoundaryRHS (real_t flux)

Add boundary right-hand side term to right hand side.

Parameters

in	<i>flux</i>	Value of flux to impose on the side
-----------	-------------	-------------------------------------

Point<real_t>& Grad (const Vect< real_t > & u)

Return gradient of a vector in element.

Parameters

in	<i>u</i>	Vector for which gradient is computed.
-----------	----------	--

virtual void setStab () [virtual], [inherited]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

int runTransient () [inherited]

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run () [inherited]

Run the equation.

If the analysis (see function setAnalysis) is STEADY_STATE, then the function solves the stationary equation.

If the analysis is TRANSIENT, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	K	Diffusion matrix
in	M	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
----	------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMatrix</code> instance
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

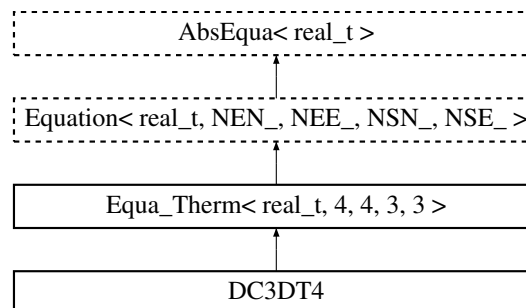
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.12 DC3DT4 Class Reference

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

Inheritance diagram for DC3DT4:



Public Member Functions

- [DC3DT4](#) ()
Default Constructor.
- [DC3DT4](#) (const [Element](#) *el)
Constructor for an element.
- [DC3DT4](#) (const [Side](#) *sd)
Constructor for a boundary side.
- [DC3DT4](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for an element (transient case).
- [DC3DT4](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time=0.)
Constructor for a boundary side (transient case).
- [DC3DT4](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for an element (transient case) with specification of time integration scheme.
- [DC3DT4](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, [real_t](#) time, [real_t](#) deltat, int scheme)
Constructor for a side (transient case) with specification of time integration scheme.
- [~DC3DT4](#) ()
Destructor.
- void [build](#) ()
Build the linear system without solving.
- void [LCapacity](#) ([real_t](#) coef=1.)
Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient coef.
- void [LCapacityToLHS](#) ([real_t](#) coef=1)
Add lumped capacity matrix to left-hand side after multiplying it by coefficient coef
- void [LCapacityToRHS](#) ([real_t](#) coef=1)
Add lumped capacity contribution to right-hand side after multiplying it by coefficient coef

- void `Capacity` (`real_t` coef=1)
Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient coef.
- void `CapacityToLHS` (`real_t` coef=1)
Add consistent capacity matrix to left-hand side after multiplying it by coefficient coef
- void `CapacityToRHS` (`real_t` coef=1)
Add consistent capacity contribution to right-hand side after multiplying it by coefficient coef
- void `Diffusion` (`real_t` coef=1)
Add diffusion matrix to left hand side after multiplying it by coefficient coef.
- void `Diffusion` (const `DMatrix`< `real_t` > &diff, `real_t` coef=1)
Add diffusion matrix to left hand side after multiplying it by coefficient coef
- void `DiffusionToRHS` (`real_t` coef=1)
Add diffusion contribution to right hand side after multiplying it by coefficient coef
- void `Convection` (`real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `Convection` (const `Point`< `real_t` > &v, `real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `Convection` (const `Vect`< `Point`< `real_t` > > &v, `real_t` coef=1)
Add convection matrix to left-hand side after multiplying it by coefficient coef
- void `RHS_Convection` (const `Point`< `real_t` > &v, `real_t` coef=1.)
Add convection contribution to right-hand side after multiplying it by coefficient coef
- void `BodyRHS` (`UserData`< `real_t` > &ud, `real_t` coef=1)
Add body right-hand side term to right hand side after multiplying it by coefficient coef
- void `BodyRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
Add body right-hand side term to right hand side.
- void `BoundaryRHS` (`UserData`< `real_t` > &ud, `real_t` coef=1)
Add boundary right-hand side term to right hand side after multiplying it by coefficient coef
- void `BoundaryRHS` (const `Vect`< `real_t` > &b, int opt=`GLOBAL_ARRAY`)
Add boundary right-hand side term to right hand side after multiplying it by coefficient coef
- void `BoundaryRHS` (`real_t` flux)
Add boundary right-hand side flux to right hand side.
- `Point`< `real_t` > `Flux` () const
Return (constant) heat flux in element.
- `Point`< `real_t` > `Grad` (const `Vect`< `real_t` > &u) const
Return gradient of vector u in element. u is a local vector.
- void `Periodic` (`real_t` coef=1.e20)
Add contribution of periodic boundary condition (by a penalty technique).
- virtual void `setStab` ()
Set stabilized formulation.
- void `setLumpedCapacity` ()
Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setCapacity` ()
Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.
- void `setDiffusion` ()
Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.

- virtual void `ConvectionToRHS` (`real_t` coef=1.)
Add convection term to right-hand side.
- void `setConvection` ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- int `runTransient` ()
Run one time step.
- int `run` ()
Run the equation.
- void `setRhoCp` (const `real_t` &rhocp)
Set product of Density by Specific heat (constants)
- void `setConductivity` (const `real_t` &diff)
Set (constant) thermal conductivity.
- void `RhoCp` (const string &exp)
Set product of Density by Specific heat given by an algebraic expression.
- void `Conduc` (const string &exp)
Set thermal conductivity given by an algebraic expression.
- void `buildEigen` (`SkSMatrix`< `real_t` > &K, `SkSMatrix`< `real_t` > &M)
Build global stiffness and mass matrices for the eigen system.
- void `buildEigen` (`SkSMatrix`< `real_t` > &K, `Vect`< `real_t` > &M)
Build global diffusion and capacity matrices for the eigen system.
- void `updateBC` (const `Element` &el, const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect`< `real_t` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector` (`Vect`< `real_t` > &b)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect`< `real_t` > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect`< `real_t` > &b)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix`< `real_t` > *A)
Assemble element matrix into global one.

- void `ElementAssembly` (`SkMatrix`< `real_t` > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix`< `real_t` > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix`< `real_t` > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix`< `real_t` > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect`< `real_t` > &v)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix`< `real_t` > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix`< `real_t` > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix`< `real_t` > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix`< `real_t` > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix`< `real_t` > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix`< `real_t` > *A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix`< `real_t` > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect`< `real_t` > &v)
Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
Assemble product of side matrix by side vector into global vector.
- `size_t` `getNbNodes` () const
Return number of element nodes.
- `size_t` `getNbEq` () const
Return number of element equations.
- `real_t` * `A` ()
Return element matrix as a C-array.
- `real_t` * `sA` ()
Return side matrix as a C-array.
- `real_t` * `b` ()
Return element right-hand side as a C-array.
- `real_t` * `sb` ()
Return side right-hand side as a C-array.

- `real_t * Prev ()`
Return element matrix as a C-array.
- `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
Return element matrix as a LocalMatrix instance.
- `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
Return side matrix as a LocalMatrix instance.
- `LocalVect< real_t, NEE_ > & Eb ()`
Return element right-hand side as a LocalVect instance.
- `LocalVect< real_t, NEE_ > & Ep ()`
Return element matrix as a C-array.
- `void setInitialSolution (const Vect< real_t > &u)`
Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`
Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`
Return reference to Mesh instance.
- `LinearSolver< real_t > & getLinearSolver ()`
Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`
Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n-th eigenvalue.
- `void getEigenvector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void setMaterial ()`
Set material properties.
- `void Init (const Element *el)`
Set element arrays to zero.
- `void Init (const Side *sd)`
Set side arrays to zero.

7.12.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 3-D domains using 4-Node tetrahedra.

Note that members calculating element arrays have as an argument a real coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.12.2 Constructor & Destructor Documentation

DC3DT4 ()

Default Constructor.

Constructs an empty equation.

DC3DT4 (const Element * el)

Constructor for an element.

Parameters

<i>in</i>	<i>el</i>	Pointer to element.
-----------	-----------	---------------------

DC3DT4 (const Side * sd)

Constructor for a boundary side.

Parameters

	<i>sd</i>	[in] Pointer to side.
--	-----------	-----------------------

DC3DT4 (const Element * el, const Vect< real_t > & u, real_t time = 0.)

Constructor for an element (transient case).

Parameters

<i>in</i>	<i>el</i>	Pointer to element.
<i>in</i>	<i>u</i>	Vect instance that contains solution at previous time step.
<i>in</i>	<i>time</i>	Current time value [Default: 0].

DC3DT4 (const Side * sd, const Vect< real_t > & u, real_t time = 0.)

Constructor for a boundary side (transient case).

Parameters

<i>in</i>	<i>sd</i>	Pointer to side.
<i>in</i>	<i>u</i>	Vect instance that contains solution at previous time step.
<i>in</i>	<i>time</i>	Current time value [Default: 0].

DC3DT4 (const Element * el, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value.
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme: <ul style="list-style-type: none"> • FORWARD_EULER: Forward Euler scheme • BACKWARD_EULER: Backward Euler scheme • CRANK_NICOLSON: Crank-Nicolson Euler scheme

DC3DT4 (`const Side * sd`, `const Vect< real_t > & u`, `real_t time`, `real_t deltat`, `int scheme`)

Constructor for a side (transient case) with specification of time integration scheme.

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value.
in	<i>deltat</i>	Value of time step
in	<i>scheme</i>	Time Integration Scheme (): <ul style="list-style-type: none"> • FORWARD_EULER: for Forward Euler scheme • BACKWARD_EULER: for Backward Euler scheme • CRANK_NICOLSON: for Crank-Nicolson Euler scheme

7.12.3 Member Function Documentation

void LCapacity (`real_t coef = 1.`)

Add lumped capacity contribution to left and right-hand sides after multiplying it by coefficient `coef`.

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
----	-------------	---

void LCapacityToLHS (`real_t coef = 1`) [virtual]

Add lumped capacity matrix to left-hand side after multiplying it by coefficient `coef`

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
----	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void LCapacityToRHS (`real_t coef = 1`) [virtual]

Add lumped capacity contribution to right-hand side after multiplying it by coefficient `coef`

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void Capacity (real_t coef = 1)

Add Consistent capacity contribution to left and right-hand sides after multiplying it by coefficient *coef*.

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

void CapacityToLHS (real_t coef = 1) [virtual]

Add consistent capacity matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void CapacityToRHS (real_t coef = 1) [virtual]

Add consistent capacity contribution to right-hand side after multiplying it by coefficient *coef*

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void Diffusion (real_t coef = 1) [virtual]

Add diffusion matrix to left hand side after multiplying it by coefficient *coef*.

Parameters

<i>in</i>	<i>coef</i>	Coefficient to multiply by added term (default value = 1).
-----------	-------------	--

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void Diffusion (const DMatrix< real_t > & diff, real_t coef = 1)

Add diffusion matrix to left hand side after multiplying it by coefficient *coef*

Case where the diffusivity matrix is given as an argument.

Parameters

<i>in</i>	<i>diff</i>	Diffusion matrix (class DMatrix).
<i>in</i>	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void DiffusionToRHS (real_t coef = 1) [virtual]

Add diffusion contribution to right hand side after multiplying it by coefficient *coef*

To be used for explicit diffusion term

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void Convection (real_t coef = 1) [virtual]

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field has been previously defined

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Therm< real_t, 4, 4, 3, 3 >](#).

void Convection (const Point< real_t > & v, real_t coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Parameters

in	<i>v</i>	Constant velocity vector
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void Convection (const Vect< Point< real_t > > & v, real_t coef = 1)

Add convection matrix to left-hand side after multiplying it by coefficient *coef*

Case where velocity field is given by a vector *v*.

Parameters

in	<i>v</i>	Velocity vector.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void RHS_Convection (const Point< real_t > & v, real_t coef = 1.)

Add convection contribution to right-hand side after multiplying it by coefficient *coef*

To be used for explicit convection term.

Parameters

in	<i>v</i>	Velocity vector.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void BodyRHS (UserData< real_t > & ud, real_t coef = 1)

Add body right-hand side term to right hand side after multiplying it by coefficient *coef*

Parameters

in	<i>ud</i>	Instance of UserData or of an inherited class. Contains a member function that provides body source.
in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].

void BodyRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>b</i>	Local vector containing source at element nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 4 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BoundaryRHS (UserData< real_t > & ud, real_t coef = 1)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Parameters

in	<i>ud</i>	Instance of UserData or of an inherited class. Contains a member function that provides body source.
in	<i>coef</i>	Value by which the added term is multiplied [Default: 1].

void BoundaryRHS (const Vect< real_t > & b, int opt = GLOBAL_ARRAY)

Add boundary right-hand side term to right hand side after multiplying it by coefficient coef

Case where body source is given by a vector

Parameters

in	<i>b</i>	Vector containing source at side nodes.
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

void BoundaryRHS (real_t flux)

Add boundary right-hand side flux to right hand side.

Parameters

in	<i>flux</i>	Vector containing source at side nodes.
-----------	-------------	---

void Periodic (real_t coef = 1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC_A on one side and PERIODIC_B on the opposite side.

Parameters

in	<i>coef</i>	Value of penalty parameter [Default: 1.e20].
-----------	-------------	--

virtual void setStab () [virtual], [inherited]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

int runTransient () [inherited]

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run () [inherited]

Run the equation.

If the analysis (see function setAnalysis) is STEADY_STATE, then the function solves the stationary equation.

If the analysis is TRANSIENT, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	<i>K</i>	Diffusion matrix
in	<i>M</i>	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
----	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<code>in</code>	<code>b</code>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------------	----------------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Element` pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Global matrix stored as an <code>SkSMatrix</code> instance
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)

out	<i>b</i>	Global vector to add (Vect instance)
-----	----------	--------------------------------------

void AxbAssembly (const Side & sd, const Vect< real.t > & x, Vect< real.t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real.t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

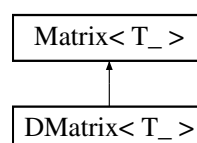
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.13 DMatrix< T_ > Class Template Reference

To handle dense matrices.

Inheritance diagram for DMatrix< T_ >:



Public Member Functions

- **DMatrix** ()
Default constructor.
- **DMatrix** (size_t nr)
Constructor for a matrix with nr rows and nr columns.
- **DMatrix** (size_t nr, size_t nc)
Constructor for a matrix with nr rows and nc columns.
- **DMatrix** (Vect< T_ > &v)
Constructor that uses a Vect instance. The class uses the memory space occupied by this vector.
- **DMatrix** (const DMatrix< T_ > &m)
Copy Constructor.
- **~DMatrix** ()
Destructor.
- void **setDiag** ()
Store diagonal entries in a separate internal vector.
- void **setSize** (size_t size)
Set size (number of rows) of matrix.
- void **setSize** (size_t nr, size_t nc)
Set size (number of rows and columns) of matrix.
- Vect< T_ > **getColumn** (size_t j) const
Get j -th column vector.
- Vect< T_ > **getRow** (size_t i) const
Get i -th row vector.
- void **set** (size_t i, size_t j, const T_ &val)
Assign a constant value to an entry of the matrix.
- void **setRow** (size_t i, const Vect< T_ > &v)
Copy a given vector to a prescribed row in the matrix.
- void **setColumn** (size_t i, const Vect< T_ > &v)
Copy a given vector to a prescribed column in the matrix.
- void **MultAdd** (T_ a, const Vect< T_ > &x, Vect< T_ > &y) const

- Multiply matrix by vector $a*x$ and add result to y .*

 - void **MultAdd** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
- Multiply matrix by vector x and add result to y .*

 - void **Mult** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
- Multiply matrix by vector x and save result in y .*

 - void **TMult** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
- Multiply transpose of matrix by vector x and add result in y .*

 - void **add** (size_t i, size_t j, const T_ &val)
- Add constant val to entry (i, j) of the matrix.*

 - void **Axpy** (T_ a, const **DMatrix**< T_ > &m)
- Add to matrix the product of a matrix by a scalar.*

 - void **Axpy** (T_ a, const **Matrix**< T_ > *m)
- Add to matrix the product of a matrix by a scalar.*

 - T_ **operator()** (size_t i, size_t j) const
- Operator () (Constant version). Return $a(i, j)$*

 - T_ & **operator()** (size_t i, size_t j)
- Operator () (Non constant version). Return $a(i, j)$*

 - int **Factor** ()
- Factorize matrix (LU factorization).*

 - int **Solve** (**Vect**< T_ > &b)
- Solve system with factorized matrix (forward and back substitution).*

 - int **Solve** (const **Vect**< T_ > &b, **Vect**< T_ > &x)
- Solve system with factorized matrix (forward and back substitution).*

 - **DMatrix** & **operator=** (**DMatrix**< T_ > &m)
- Operator =*

 - **DMatrix** & **operator+=** (const **DMatrix**< T_ > &m)
- Operator +=.*

 - **DMatrix** & **operator-=** (const **DMatrix**< T_ > &m)
- Operator -=.*

 - **DMatrix** & **operator=** (const T_ &x)
- Operator =*

 - **DMatrix** & **operator*=** (const T_ &x)
- Operator *=*

 - **DMatrix** & **operator+=** (const T_ &x)
- Operator +=*

 - **DMatrix** & **operator-=** (const T_ &x)
- Operator -=*

 - size_t **getColInd** (size_t i) const
- Return storage information.*

 - T_ * **getArray** () const
- Return matrix as C-Array.*

 - T_ **get** (size_t i, size_t j) const
- Return entry (i, j) of matrix.*

 - void **setPrintView** (size_t rmin, size_t rmax, size_t cmin, size_t cmax)
- Set a window for matrix printing.*

 - size_t **getNbRows** () const

- Return number of rows.*

 - size_t `getNbColumns` () const
- Return number of columns.*

 - void `setPenal` (real_t p)

Set Penalty Parameter (For boundary condition prescription).
- void `setDiagonal` ()

Set the matrix as diagonal.
- void `setDiagonal` (const class `Mesh` &mesh)

Initialize matrix storage in the case where only diagonal terms are stored.
- T_ `getDiag` (size_t k) const

Return k-th diagonal entry of matrix.
- size_t `size` () const

Return matrix dimension (Number of rows and columns).
- void `Assembly` (const class `Element` *el, T_ *a)

Assembly of element matrix into global matrix.
- void `Assembly` (const class `Element` *el, const `DMatrix`< T_ > &a)

Assembly of element matrix into global matrix.
- void `Assembly` (const class `Side` *sd, T_ *a)

Assembly of side matrix into global matrix.
- void `Assembly` (const class `Side` *sd, const `DMatrix`< T_ > &a)

Assembly of side matrix into global matrix.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition.
- void `Prescribe` (int dof, int code, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- void `Prescribe` (`Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, int flag=0)

Impose by a penalty method a homogeneous (=0) essential boundary condition.
- void `Prescribe` (`Vect`< T_ > &b, int flag=0)

Impose by a penalty method a homogeneous (=0) essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- int `FactorAndSolve` (`Vect`< T_ > &b)

Factorize matrix and solve the linear system.
- int `FactorAndSolve` (const `Vect`< T_ > &b, `Vect`< T_ > &x)

Factorize matrix and solve the linear system.

- unsigned long `getLength` () const
Return number of stored terms in matrix.
- int `isDiagonal` () const
Say if matrix is diagonal or not.
- int `isFactorized` () const
Say if matrix is factorized or not.
- virtual size_t `getRowPtr` (size_t i) const
Return Row pointer for row i (See the description for class `SpMatrix`).
- T_ `operator`() (size_t i) const
Operator () with one argument (Constant version).
- T_ & `operator`() (size_t i)
Operator () with one argument (Non Constant version).
- T_ & `operator`[] (size_t k)
Operator [] (Non constant version).
- T_ `operator`[] (size_t k) const
Operator [] (Constant version).
- `Matrix` & `operator+=` (const `Matrix`< T_ > &m)
Operator +=.
- `Matrix` & `operator-=` (const `Matrix`< T_ > &m)
Operator -=.
- void `getPrintView` (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
Return window data for matrix printing.

7.13.1 Detailed Description

`template<class T_>class OFELI::DMatrix< T_ >`

To handle dense matrices.

This class enables storing and manipulating general dense matrices. Matrices can be square or rectangle ones.

Template Parameters

<code>< T_ ></code>	Data type (real_t, float, complex<real_t>, ...)
---------------------------	---

7.13.2 Constructor & Destructor Documentation

`DMatrix ()`

Default constructor.

Initializes a zero-dimension matrix.

`DMatrix (size_t nr)`

Constructor for a matrix with `nr` rows and `nr` columns.

Matrix entries are set to 0.

`DMatrix (size_t nr, size_t nc)`

Constructor for a matrix with `nr` rows and `nc` columns.

`Matrix` entries are set to 0.

DMatrix (Vect< T_ > & v)

Constructor that uses a [Vect](#) instance. The class uses the memory space occupied by this vector.

Parameters

<i>in</i>	<i>v</i>	Vector to copy
-----------	----------	----------------

DMatrix (const DMatrix< T_ > & m)

Copy Constructor.

Parameters

<i>in</i>	<i>m</i>	Matrix to copy
-----------	----------	----------------

7.13.3 Member Function Documentation

void setSize (size_t size)

Set size (number of rows) of matrix.

Parameters

<i>in</i>	<i>size</i>	Number of rows and columns.
-----------	-------------	-----------------------------

void setSize (size_t nr, size_t nc)

Set size (number of rows and columns) of matrix.

Parameters

<i>in</i>	<i>nr</i>	Number of rows.
<i>in</i>	<i>nc</i>	Number of columns.

void set (size_t i, size_t j, const T_ & val) [virtual]

Assign a constant value to an entry of the matrix.

Parameters

<i>in</i>	<i>i</i>	row index of matrix
<i>in</i>	<i>j</i>	column index of matrix
<i>in</i>	<i>val</i>	Value to assign to a(i,j).

Implements [Matrix< T_ >](#).**void setRow (size_t i, const Vect< T_ > & v)**

Copy a given vector to a prescribed row in the matrix.

Parameters

<i>in</i>	<i>i</i>	row index to be assigned
<i>in</i>	<i>v</i>	Vector to copy

void setColumn (size_t i, const Vect< T_ > & v)

Copy a given vector to a prescribed column in the matrix.

Parameters

in	<i>i</i>	column index to be assigned
in	<i>v</i>	Vector to copy

void MultAdd (T_ a, const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector a*x and add result to y.

Parameters

in	<i>a</i>	constant to multiply by
in	<i>x</i>	Vector to multiply by a
in,out	<i>y</i>	on input, vector to add to. On output, result.

Implements [Matrix< T_ >](#).

void MultAdd (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and add result to y.

Parameters

in	<i>x</i>	Vector to add to y
in,out	<i>y</i>	on input, vector to add to. On output, result.

Implements [Matrix< T_ >](#).

void Mult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and save result in y.

Parameters

in	<i>x</i>	Vector to add to y
out	<i>y</i>	Result.

Implements [Matrix< T_ >](#).

void TMult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply transpose of matrix by vector x and add result in y.

Parameters

in	<i>x</i>	Vector to add to y
in,out	<i>y</i>	on input, vector to add to. On output, result.

Implements [Matrix< T_ >](#).

void add (size_t i, size_t j, const T_ & val) [virtual]

Add constant val to entry (i,j) of the matrix.

Parameters

in	<i>i</i>	row index
in	<i>j</i>	column index
in	<i>val</i>	Constant to add

Implements [Matrix< T_ >](#).

void Axy (T_ a, const DMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Matrix by which <i>a</i> is multiplied. The result is added to current instance

void Apxy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Matrix by which <i>a</i> is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

T_ operator() (size_t i, size_t j) const [virtual]

Operator () (Constant version). Return *a*(*i*,*j*)

Parameters

in	<i>i</i>	row index
in	<i>j</i>	column index

Implements [Matrix< T_ >](#).

T_ & operator() (size_t i, size_t j) [virtual]

Operator () (Non constant version). Return *a*(*i*,*j*)

Parameters

in	<i>i</i>	row index
in	<i>j</i>	column index

Implements [Matrix< T_ >](#).

int Factor () [virtual]

Factorize matrix (LU factorization).

Returns

- 0 if factorization was normally performed,
- *n* if the *n*-th pivot is null.

Implements [Matrix< T_ >](#).

int Solve (Vect< T_ > & b) [virtual]

Solve system with factorized matrix (forward and back substitution).

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output.
--------	----------	--

Returns

- 0 if solution was normally performed,
 - *n* if the *n*-th pivot is null.
- Solution is performed only if factorization has previously been invoked.

Implements [Matrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve system with factorized matrix (forward and back substitution).

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is null

Solution is performed only if factorization has previously been invoked.

DMatrix< T_ > & operator= (DMatrix< T_ > & m)

Operator =

Copy matrix *m* to current matrix instance.**DMatrix< T_ > & operator+= (const DMatrix< T_ > & m)**

Operator +=.

Add matrix *m* to current matrix instance.**DMatrix< T_ > & operator-= (const DMatrix< T_ > & m)**

Operator -=.

Subtract matrix *m* from current matrix instance.**DMatrix< T_ > & operator= (const T_ & x)**

Operator =

Assign matrix to identity times *x***DMatrix< T_ > & operator*= (const T_ & x)**

Operator *=

Premultiply matrix entries by constant value *x*.**DMatrix< T_ > & operator+= (const T_ & x)**

Operator +=

Add constant value *x* to matrix entries**DMatrix< T_ > & operator-= (const T_ & x)**

Operator -=

Subtract constant value *x* from matrix entries.**size_t getColInd (size_t i) const** [virtual]

Return storage information.

Parameters

in	<i>i</i>	Index of stored entry in matrix
-----------	----------	---------------------------------

Returns

Column index for this entry

Reimplemented from [Matrix< T_ >](#).

T_* [getArray](#) () const

Return matrix as C-Array.

Matrix is stored row by row.

void [setPrintView](#) (size_t rmin, size_t rmax, size_t cmin, size_t cmax)

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void [setDiagonal](#) (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ [getDiag](#) (size_t k) const [inherited]

Return k-th diagonal entry of matrix.

First entry is given by [getDiag\(1\)](#).

void [Assembly](#) (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void [Assembly](#) (const class Element * el, const [DMatrix](#)< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void [Assembly](#) (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This gunction is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
--------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix `m` to current matrix instance.

Matrix< T_ > & operator-= (const Matrix< T_ > & m) [inherited]

Operator -=.

Subtract matrix `m` from current matrix instance.

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const
[inherited]

Return window data for matrix printing.

Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.14 Domain Class Reference

To store and treat finite element geometric information.

Public Member Functions

- [Domain](#) ()
Constructor of a null domain.
- [Domain](#) (const string &file)
Constructor with an input file.
- [~Domain](#) ()
Destructor.
- void [setFile](#) (string file)
Set file containing [Domain](#) data.
- void [setDim](#) (size_t d)
Set space dimension.
- size_t [getDim](#) () const
Return space dimension.
- void [setNbDOF](#) (size_t n)
Set number of degrees of freedom.
- size_t [getNbDOF](#) () const
Return number of degrees of freedom.
- size_t [getNbVertices](#) () const
Return number of vertices.
- size_t [getNbLines](#) () const
Return number of lines.
- size_t [getNbContours](#) () const
Return number of contours.
- size_t [getNbHoles](#) () const

- Return number of holes.*
- `size_t getNbSubDomains ()` const
 - Return number of sub-domains.*
- `int get ()`
 - Read domain data interactively.*
- `void get (const string &file)`
 - Read domain data from a data file.*
- `Mesh & getMesh ()` const
 - Return reference to generated *Mesh* instance.*
- `void genGeo (string file)`
 - Generate geometry file.*
- `void genMesh ()`
 - Generate 2-D mesh.*
- `void genMesh (const string &file)`
 - Generate 2-D mesh and save in file (*OFELI* format)*
- `void genMesh (string geo_file, string bamg_file, string mesh_file)`
 - Generate 2-D mesh and save geo, bamg and mesh file (*OFELI* format)*
- `void generateMesh (const string &file)`
 - Generate 2-D mesh using the BAMG mesh generator.*
- `Domain & operator*= (real_t a)`
 - Operator *=*
- `void insertVertex (real_t x, real_t y, real_t h, int code)`
 - Insert a vertex.*
- `void insertLine (size_t n1, size_t n2, int dc, int nc)`
 - Insert a straight line.*
- `void insertCircle (size_t n1, size_t n2, size_t n3, int dc, int nc)`
 - Insert a circular arc.*
- `void insertRequiredVertex (size_t v)`
 - Insert a required (imposed) vertex.*
- `void insertRequiredEdge (size_t e)`
 - Insert a required (imposed) edge (or line)*
- `void insertSubDomain (size_t n, int code)`
 - Insert subdomain.*
- `void insertSubDomain (size_t ln, int orient, int code)`
 - Insert subdomain.*
- `Point< real_t > getMinCoord ()` const
 - Return minimum coordinates of vertices.*
- `Point< real_t > getMaxCoord ()` const
 - Return maximum coordinates of vertices.*
- `real_t getMinh ()` const
 - Return minimal value of mesh size.*

7.14.1 Detailed Description

To store and treat finite element geometric information.

This class is essentially useful to construct data for mesh generators.

7.14.2 Constructor & Destructor Documentation

Domain ()

Constructor of a null domain.

This constructor assigns maximal values of parameters.

Domain (const string & file)

Constructor with an input file.

Parameters

<i>in</i>	<i>file</i>	Input file in the XML format defining the domain
-----------	-------------	--

7.14.3 Member Function Documentation

void get (const string & file)

Read domain data from a data file.

Parameters

<i>in</i>	<i>file</i>	Input file in Domain XML format
-----------	-------------	---

void genMesh (const string & file)

Generate 2-D mesh and save in file ([OFELI](#) format)

Parameters

<i>in</i>	<i>file</i>	File where the generated mesh is saved
-----------	-------------	--

void genMesh (string geo_file, string bamg_file, string mesh_file)

Generate 2-D mesh and save geo, bamg and mesh file ([OFELI](#) format)

Parameters

<i>in</i>	<i>geo_file</i>	Geo file
<i>in</i>	<i>bamg_file</i>	Bamg file
<i>in</i>	<i>mesh_file</i>	File where the generated mesh is saved

void generateMesh (const string & file)

Generate 2-D mesh using the BAMG mesh generator.

Parameters

<i>in</i>	<i>file</i>	that contains output mesh after execution
-----------	-------------	---

Domain& operator*= (real_t a)

Operator *=

Rescale domain coordinates by multiplying by a factor

Parameters

in	<i>a</i>	Value to multiply by
-----------	----------	----------------------

void insertVertex (real_t x, real_t y, real_t h, int code)

Insert a vertex.

Parameters

in	<i>x</i>	x-coordinate of vertex
in	<i>y</i>	y-coordinate of vertex
in	<i>h</i>	mesh size around vertex
in	<i>code</i>	code of coordinate

void insertLine (size_t n1, size_t n2, int dc, int nc)

Insert a straight line.

Parameters

in	<i>n1</i>	Label of the first vertex of line
in	<i>n2</i>	Label of the second vertex of line
in	<i>dc</i>	Code to associate to created nodes (Dirichlet)
in	<i>nc</i>	Code to associate to line (Neumann)

void insertCircle (size_t n1, size_t n2, size_t n3, int dc, int nc)

Insert a circular arc.

Parameters

in	<i>n1</i>	Label of vertex defining the first end of the arc
in	<i>n2</i>	Label of vertex defining the second end of the arc
in	<i>n3</i>	Label of vertex defining the center of the arc
in	<i>dc</i>	Dirichlet code for nodes on the arc
in	<i>nc</i>	Neumann code for sides on the arc

void insertRequiredVertex (size_t v)

Insert a required (imposed) vertex.

Parameters

in	<i>v</i>	Label of vertex
-----------	----------	-----------------

void insertRequiredEdge (size_t e)

Insert a required (imposed) edge (or line)

Parameters

in	<i>e</i>	Label of line
-----------	----------	---------------

void insertSubDomain (size_t n, int code)

Insert subdomain.

Parameters

<code>in</code>	<code>n</code>	
<code>in</code>	<code>code</code>	

void insertSubDomain (size_t In, int orient, int code)

Insert subdomain.

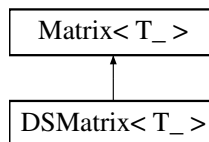
Parameters

<code>in</code>	<code>In</code>	Line label
<code>in</code>	<code>orient</code>	Orientation (1 or -1)
<code>in</code>	<code>code</code>	Subdomain code or reference

7.15 DMatrix< T_ > Class Template Reference

To handle symmetric dense matrices.

Inheritance diagram for DMatrix< T_ >:



Public Member Functions

- [DMatrix](#) ()
Default constructor.
- [DMatrix](#) (size_t dim)
Constructor that for a symmetric matrix with given number of rows.
- [DMatrix](#) (const [DMatrix](#)< T_ > &m)
Copy Constructor.
- [~DMatrix](#) ()
Destructor.
- void [setDiag](#) ()
Store diagonal entries in a separate internal vector.
- void [setSize](#) (size_t dim)
Set size (number of rows) of matrix.
- void [set](#) (size_t i, size_t j, const T_ &val)
Assign constant to entry (i, j) of the matrix.
- void [add](#) (size_t i, size_t j, const T_ &val)
Add constant to an entry of the matrix.
- T_ [operator\(\)](#) (size_t i, size_t j) const
Operator () (Constant version).
- T_ & [operator\(\)](#) (size_t i, size_t j)
Operator () (Non constant version).
- [DMatrix](#)< T_ > & [operator=](#) (const [DMatrix](#)< T_ > &m)
Operator = Copy matrix m to current matrix instance.

- `DSMatrix< T_ > & operator= (const T_ &x)`
Operator = Assign matrix to identity times x.
- `int Factor ()`
Factorize matrix (LDL^T)
- `void MultAdd (const Vect< T_ > &x, Vect< T_ > &y) const`
*Multiply matrix by vector $a*x$ and add result to y.*
- `void MultAdd (T_ a, const Vect< T_ > &x, Vect< T_ > &y) const`
*Multiply matrix by vector $a*x$ and add to y.*
- `void Mult (const Vect< T_ > &x, Vect< T_ > &y) const`
Multiply matrix by vector x and save result in y.
- `void TMult (const Vect< T_ > &x, Vect< T_ > &y) const`
Multiply transpose of matrix by vector x and add result in y.
- `void Axy (T_ a, const DSMatrix< T_ > &m)`
Add to matrix the product of a matrix by a scalar.
- `void Axy (T_ a, const Matrix< T_ > *m)`
Add to matrix the product of a matrix by a scalar.
- `int Solve (Vect< T_ > &b)`
Solve system with factored matrix (forward and back substitution).
- `int Solve (const Vect< T_ > &b, Vect< T_ > &x)`
Solve system with factorized matrix (forward and back substitution).
- `T_ * getArray () const`
Return matrix as C-Array. Matrix is stored row by row. Only lower triangle is stored.
- `T_ get (size_t i, size_t j) const`
Return entry (i, j) of matrix.
- `void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)`
Set a window for matrix printing.
- `size_t getNbRows () const`
Return number of rows.
- `size_t getNbColumns () const`
Return number of columns.
- `void setPenal (real_t p)`
Set Penalty Parameter (For boundary condition prescription).
- `void setDiagonal ()`
Set the matrix as diagonal.
- `void setDiagonal (const class Mesh &mesh)`
Initialize matrix storage in the case where only diagonal terms are stored.
- `T_ getDiag (size_t k) const`
Return k-th diagonal entry of matrix.
- `size_t size () const`
Return matrix dimension (Number of rows and columns).
- `void Assembly (const class Element *el, T_ *a)`
Assembly of element matrix into global matrix.
- `void Assembly (const class Element *el, const DMatrix< T_ > &a)`
Assembly of element matrix into global matrix.
- `void Assembly (const class Side *sd, T_ *a)`
Assembly of side matrix into global matrix.

- void `Assembly` (const class `Side` *sd, const `DMatrix`< T_ > &a)
Assembly of side matrix into global matrix.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition.
- void `Prescribe` (int dof, int code, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- void `Prescribe` (`Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition.
- void `Prescribe` (`Vect`< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition, using the `Mesh` instance provided by the constructor.
- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- int `FactorAndSolve` (`Vect`< T_ > &b)
Factorize matrix and solve the linear system.
- int `FactorAndSolve` (const `Vect`< T_ > &b, `Vect`< T_ > &x)
Factorize matrix and solve the linear system.
- unsigned long `getLength` () const
Return number of stored terms in matrix.
- int `isDiagonal` () const
Say if matrix is diagonal or not.
- int `isFactorized` () const
Say if matrix is factorized or not.
- virtual size_t `getColInd` (size_t i) const
Return Column index for column i (See the description for class `SpMatrix`).
- virtual size_t `getRowPtr` (size_t i) const
Return Row pointer for row i (See the description for class `SpMatrix`).
- T_ `operator`() (size_t i) const
Operator () with one argument (Constant version).
- T_ & `operator`() (size_t i)
Operator () with one argument (Non Constant version).
- T_ & `operator`[] (size_t k)
Operator [] (Non constant version).
- T_ `operator`[] (size_t k) const
Operator [] (Constant version).

- **Matrix** & **operator+=** (const **Matrix**< T_ > &m)
Operator +=.
- **Matrix** & **operator+=** (const T_ &x)
Operator +=.
- **Matrix** & **operator-=** (const **Matrix**< T_ > &m)
Operator -=.
- **Matrix** & **operator-=** (const T_ &x)
Operator -=.
- **Matrix** & **operator*=** (const T_ &x)
*Operator *=.*
- void **getPrintView** (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
Return window data for matrix printing.

7.15.1 Detailed Description

template<class T_>class OFELI::DSMatrix< T_ >

To handle symmetric dense matrices.

This class enables storing and manipulating symmetric dense matrices.

Template Parameters

< T_ >	Data type (double, float, complex<double>, ...)
--------	---

7.15.2 Constructor & Destructor Documentation

DSMatrix (size_t dim)

Constructor that for a symmetric matrix with given number of rows.

Parameters

in	<i>dim</i>	Number of rows
----	------------	----------------

DSMatrix (const DSMatrix< T_ > & m)

Copy Constructor.

Parameters

in	<i>m</i>	DSMatrix instance to copy
----	----------	---

7.15.3 Member Function Documentation

void setSize (size_t dim)

Set size (number of rows) of matrix.

Parameters

in	<i>dim</i>	Number of rows and columns.
----	------------	-----------------------------

void set (size_t i, size_t j, const T_ & val) [virtual]

Assign constant to entry (i,j) of the matrix.

Parameters

in	<i>i</i>	row index
in	<i>j</i>	column index
in	<i>val</i>	value to assign to a(i,j)

Implements [Matrix< T_ >](#).

void add (size_t i, size_t j, const T_ & val) [virtual]

Add constant to an entry of the matrix.

Parameters

in	<i>i</i>	row index
in	<i>j</i>	column index
in	<i>val</i>	value to add to a(i,j)

Implements [Matrix< T_ >](#).

T_ operator() (size_t i, size_t j) const [virtual]

Operator () (Constant version).

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).

T_ & operator() (size_t i, size_t j) [virtual]

Operator () (Non constant version).

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).

int Factor () [virtual]

Factorize matrix (LDL^T)

Returns

- 0, if factorization was normally performed,
- n, if the n-th pivot is null.

Implements [Matrix< T_ >](#).

void MultAdd (T_ a, const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector $a*x$ and add to y .

Parameters

in	<i>a</i>	Constant to multiply by matrix
----	----------	--------------------------------

in	x	Vector to multiply by matrix
in,out	y	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void TMult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply transpose of matrix by vector x and add result in y.

Parameters

in	x	Vector to add to y
in,out	y	on input, vector to add to. On output, result.

Implements [Matrix< T_ >](#).

void Apxy (T_ a, const DSMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

in	a	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

void Apxy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	a	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

int Solve (Vect< T_ > & b) [virtual]

Solve system with factored matrix (forward and back substitution).

Parameters

in	b	Vect instance that contains right-hand side on input and solution on output.
----	---	--

Returns

- 0, if solution was normally performed,
- n, if the n-th pivot is null.

Solution is performed only if factorization has previously been invoked.

Implements [Matrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve system with factorized matrix (forward and back substitution).

Parameters

in	b	Vect instance that contains right-hand side
in	x	Vect instance that contains solution

Returns

- 0, if solution was normally performed,
- n , if the n -th pivot is null.

Solution is performed only if factorization has previously been invoked.

void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return k -th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function [setPenal](#)(..).

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function [setPenal](#)(..).

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function [setPenal](#)(..).

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This gunction is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
--------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.

Matrix< T_ > & operator+= (const T_ & x) [inherited]

Operator +=.

Add constant value *x* to all matrix entries.

Matrix< T_ > & operator-= (const Matrix< T_ > & m) [inherited]

Operator -=.

Subtract matrix *m* from current matrix instance.

Matrix< T_ > & operator-= (const T_ & x) [inherited]

Operator -=.

Subtract constant value *x* from all matrix entries.

Matrix< T_ > & operator*= (const T_ & x) [inherited]

Operator * =.

Premultiply matrix entries by constant value *x*

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const
[inherited]

Return window data for matrix printing.

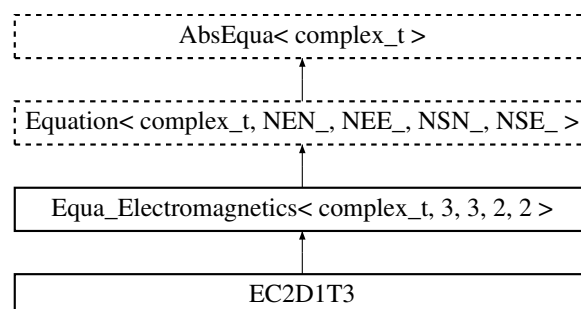
Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.16 EC2D1T3 Class Reference

Eddy current problems in 2-D domains using solenoidal approximation.

Inheritance diagram for EC2D1T3:



Public Member Functions

- **EC2D1T3** ()
Default constructor.
- **EC2D1T3** (const **Element** *el)
Constructor using element data.
- **EC2D1T3** (const **Side** *side)
Constructor using side data.
- **EC2D1T3** (const **Element** *el, const **Vect**< **complex.t** > &u, const **real.t** &time=0.)
Constructor using element and previous time data.
- **EC2D1T3** (const **Side** *sd, const **Vect**< **complex.t** > &u, const **real.t** &time=0.)
Constructor using side and previous time data.
- **~EC2D1T3** ()
Destructor.
- void **Magnetic** (**real.t** omega, **real.t** coef=1.)
Add magnetic contribution to matrix.
- void **Electric** (**real.t** coef=1.)
Add electric contribution to matrix.
- **real.t** **Joule** ()
Compute Joule density in element.
- **complex.t** **IntegMF** ()
Add element integral contribution.
- **complex.t** **IntegND** (const **Vect**< **complex.t** > &h, int opt=GLOBAL_ARRAY)
Compute integral of normal derivative on edge.
- **real.t** **VacuumArea** ()
Add contribution to vacuum area calculation.
- void **updateBC** (const **Element** &el, const **Vect**< **complex.t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **updateBC** (const **Vect**< **complex.t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **DiagBC** (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void **LocalNodeVector** (**Vect**< **complex.t** > &b)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **complex.t** > &b, **LocalVect**< **complex.t**, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **complex.t** > &b, **LocalVect**< **complex.t**, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void **ElementNodeVectorSingleDOF** (const **Vect**< **complex.t** > &b, **LocalVect**< **complex.t**, NE-N_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementSideVector** (const **Vect**< **complex.t** > &b, **LocalVect**< **complex.t**, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementVector** (const **Vect**< **complex.t** > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void **SideVector** (const **Vect**< **complex.t** > &b)

- Localize Side Vector.*
- void `ElementNodeCoordinates ()`
 - Localize coordinates of element nodes.*
- void `SideNodeCoordinates ()`
 - Localize coordinates of side nodes.*
- void `ElementAssembly (Matrix< complex_t > *A)`
 - Assemble element matrix into global one.*
- void `ElementAssembly (SkSMatrix< complex_t > &A)`
 - Assemble element matrix into global one.*
- void `ElementAssembly (SkMatrix< complex_t > &A)`
 - Assemble element matrix into global one.*
- void `ElementAssembly (SpMatrix< complex_t > &A)`
 - Assemble element matrix into global one.*
- void `ElementAssembly (TrMatrix< complex_t > &A)`
 - Assemble element matrix into global one.*
- void `ElementAssembly (Vect< complex_t > &v)`
 - Assemble element vector into global one.*
- void `DGElementAssembly (Matrix< complex_t > *A)`
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly (SkSMatrix< complex_t > &A)`
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly (SkMatrix< complex_t > &A)`
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly (SpMatrix< complex_t > &A)`
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly (TrMatrix< complex_t > &A)`
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `SideAssembly (Matrix< complex_t > *A)`
 - Assemble side (edge or face) matrix into global one.*
- void `SideAssembly (SkSMatrix< complex_t > &A)`
 - Assemble side (edge or face) matrix into global one.*
- void `SideAssembly (SkMatrix< complex_t > &A)`
 - Assemble side (edge or face) matrix into global one.*
- void `SideAssembly (SpMatrix< complex_t > &A)`
 - Assemble side (edge or face) matrix into global one.*
- void `SideAssembly (Vect< complex_t > &v)`
 - Assemble side (edge or face) vector into global one.*
- void `AxbAssembly (const Element &el, const Vect< complex_t > &x, Vect< complex_t > &b)`
 - Assemble product of element matrix by element vector into global vector.*
- void `AxbAssembly (const Side &sd, const Vect< complex_t > &x, Vect< complex_t > &b)`
 - Assemble product of side matrix by side vector into global vector.*
- size_t `getNbNodes ()` const
 - Return number of element nodes.*
- size_t `getNbEq ()` const
 - Return number of element equations.*
- `complex_t * A ()`

- Return element matrix as a C-array.*

 - `complex_t * sA ()`
- Return side matrix as a C-array.*

 - `complex_t * b ()`
- Return element right-hand side as a C-array.*

 - `complex_t * sb ()`
- Return side right-hand side as a C-array.*

 - `complex_t * Prev ()`
- Return element matrix as a C-array.*

 - `LocalMatrix< complex_t, NEE_, NEE_ > & EA ()`
- Return element matrix as a LocalMatrix instance.*

 - `LocalMatrix< complex_t, NSE_, NSE_ > & SA ()`
- Return side matrix as a LocalMatrix instance.*

 - `LocalVect< complex_t, NEE_ > & Eb ()`
- Return element right-hand side as a LocalVect instance.*

 - `LocalVect< complex_t, NEE_ > & Ep ()`
- Return element matrix as a C-array.*

 - `void setInitialSolution (const Vect< complex_t > &u)`
- Set initial solution (previous time step)*

 - `real_t setMaterialProperty (const string &exp, const string &prop)`
- Define a material property by an algebraic expression.*

 - `void setMesh (class Mesh &m)`
- Define mesh and renumber DOFs after removing imposed ones.*

 - `Mesh & getMesh () const`
- Return reference to Mesh instance.*

 - `LinearSolver< complex_t > & getLinearSolver ()`
- Return reference to linear solver instance.*

 - `void setSolver (int ls, int pc=IDENT_PREC)`
- Choose solver for the linear system.*

 - `int solveEigenProblem (int nb_eigv, bool g=false)`
- Compute eigenvalues and eigenvectors.*

 - `real_t getEigenValue (int n) const`
- Return the n-th eigenvalue.*

 - `void getEigenVector (int n, Vect< real_t > &v) const`
- Store the eigenvector corresponding to a given eigenvalue.*

 - `class Eigen & getEigenSolver ()`
- Return reference to eigenproblem solver.*

Protected Member Functions

- `void MagneticPermeability (const real_t &mu)`

Set (constant) magnetic permeability.
- `void MagneticPermeability (const string &exp)`

Set magnetic permeability given by an algebraic expression.
- `void ElectricConductivity (const real_t &sigma)`

- Set (constant) electric conductivity.
 - void [ElectricConductivity](#) (const string &exp)
 set electric conductivity given by an algebraic expression
- void [ElectricResistivity](#) (const [real_t](#) &rho)
 Set (constant) electric resistivity.
 - void [ElectricResistivity](#) (const string &exp)
 Set electric resistivity given by an algebraic expression.
- void [setMaterial](#) ()
 Set material properties.
 - void [Init](#) (const [Element](#) *el)
 Set element arrays to zero.
 - void [Init](#) (const [Side](#) *sd)
 Set side arrays to zero.

7.16.1 Detailed Description

Eddy current problems in 2-D domains using solenoidal approximation.

Builds finite element arrays for time harmonic eddy current problems in 2-D domains with solenoidal configurations (Magnetic field has only one nonzero component). Magnetic field is constant in the vacuum, and then zero in the outer vacuum.

Uses 3-Node triangles.

The unknown is the time-harmonic magnetic induction (complex valued).

7.16.2 Constructor & Destructor Documentation

EC2D1T3 (const [Element](#) * el)

Constructor using element data.

Parameters

in	el	Pointer to Element instance
--------------------	--------------------	---

EC2D1T3 (const [Side](#) * side)

Constructor using side data.

Parameters

in	side	Pointer to Side instance
--------------------	----------------------	--

EC2D1T3 (const [Element](#) * el, const [Vect](#)< [complex_t](#) > & u, const [real_t](#) & time = 0.)

Constructor using element and previous time data.

Parameters

in	el	Pointer to Element instance
in	u	Solution at previous iteration
in	time	Time value [Default: 0]

EC2D1T3 (const [Side](#) * sd, const [Vect](#)< [complex_t](#) > & u, const [real_t](#) & time = 0.)

Constructor using side and previous time data.

Parameters

in	<i>sd</i>	Pointer to Side instance
in	<i>u</i>	Solution at previous iteration
in	<i>time</i>	Time value [Default: 0]

7.16.3 Member Function Documentation

void Magnetic (real.t omega, real.t coef = 1.)

Add magnetic contribution to matrix.

Parameters

in	<i>omega</i>	Angular frequency
in	<i>coef</i>	Coefficient to multiply by [Default: 1]

void Electric (real.t coef = 1.)

Add electric contribution to matrix.

Parameters

in	<i>coef</i>	Coefficient to multiply by [Default: 1]
----	-------------	---

complex.t IntegND (const Vect< complex.t > & h, int opt = GLOBAL_ARRAY)

Compute integral of normal derivative on edge.

Parameters

in	<i>h</i>	Vect instance containing magnetic field at element nodes
in	<i>opt</i>	Vector <i>h</i> is local (LOCAL_ARRAY) with size 3 or global (GLOBAL_ARRAY) with size = Number of nodes [Default: GLOBAL_ARRAY].

Note

This member function is to be called within each element, it detects boundary sides as the ones with nonzero code

void updateBC (const Element & el, const Vect< complex.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< complex.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
----	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< complex_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< complex_t > & b, LocalVect< complex_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< complex_t > & b, LocalVect< complex_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< complex_t > & b, LocalVect<  
complex_t , NEN_ > & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< complex_t > & b, LocalVect< complex_t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< complex_t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< complex_t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Element` pointer `_theElement`

void ElementAssembly (Matrix< complex_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMMatrix</code> instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< complex_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< complex_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< complex_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< complex_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< complex_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< complex_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< complex_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< complex_t > & x, Vect< complex_t > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< complex_t > & x, Vect< complex_t > & b) [inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
----	-----------	---

<code>in</code>	<code>pc</code>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • <code>IDENT_PREC</code>, Identity preconditioner (no preconditioning [default]) • <code>DIAG_PREC</code>, Diagonal preconditioner • <code>ILU_PREC</code>, Incomplete LU factorization preconditioner
-----------------	-----------------	---

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

<code>in</code>	<code>nb_eigv</code>	Number of eigenvalues to compute
<code>in</code>	<code>g</code>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function `getEigen` was called with an argument `nb_eigv` greater or equal to `n`. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

Parameters

<code>in</code>	<code>n</code>	Label of the eigenvalue
<code>out</code>	<code>v</code>	Vect instance containing the corresponding eigenvector. This vector is resized.

7.17 Edge Class Reference

To describe an edge.

Public Member Functions

- [Edge](#) ()
Default Constructor.
- [Edge](#) (size_t label)
Constructor with label.
- [Edge](#) (const [Edge](#) &ed)
Copy constructor.
- [~Edge](#) ()

Destructor.

- void `Add (Node *node)`
Insert a node at end of list of nodes of edge.
- void `setLabel (size_t i)`
Assign label of edge.
- void `setFirstDOF (size_t n)`
Define First DOF.
- void `setNbDOF (size_t nb_dof)`
Define number of DOF of edge.
- void `DOF (size_t i, size_t dof)`
Define label of DOF.
- void `setDOF (size_t &first_dof, size_t nb_dof)`
Define number of DOF.
- void `setCode (size_t dof, int code)`
Assign code code to DOF number dof.
- void `AddNeighbor (Side *sd)`
Add side pointed by sd to list of edge sides.
- size_t `getLabel ()` const
Return label of edge.
- size_t `n ()` const
Return label of edge.
- size_t `getNbEq ()` const
Return number of edge equations.
- size_t `getNbDOF ()` const
Return number of DOF.
- int `getCode (size_t dof=1)` const
Return code for a given DOF of node.
- size_t `getDOF (size_t i)` const
Return label of i-th DOF.
- size_t `getFirstDOF ()` const
Return number of first dof of node.
- `Node * getPtrNode (size_t i)` const
List of element nodes.
- `Node * operator() (size_t i)` const
Operator ().
- size_t `getNodeLabel (size_t i)` const
Return node label.
- `Side * getNeighborSide (size_t i)` const
Return pointer to neighbor i-th side.
- int `isOnBoundary ()` const
Say if current edge is a boundary edge or not.
- void `setOnBoundary ()`
Say that the edge is on the boundary.
- `Node * operator() (size_t i)`
Operator ().

7.17.1 Detailed Description

To describe an edge.

Defines an edge of a 3-D finite element mesh. The edges are given in particular by a list of nodes. Each node can be accessed by the member function `getPtrNode`.

7.17.2 Constructor & Destructor Documentation

Edge ()

Default Constructor.

Initializes data to zero

Edge (size_t label)

Constructor with label.

Define an edge by giving its `label`

7.17.3 Member Function Documentation

void DOF (size_t i, size_t dof)

Define label of DOF.

Parameters

<code>in</code>	<code>i</code>	DOF index
<code>in</code>	<code>dof</code>	Its label

void setDOF (size_t & first_dof, size_t nb_dof)

Define number of DOF.

Parameters

<code>in,out</code>	<code>first_dof</code>	Label of the first DOF in input that is actualized
<code>in</code>	<code>nb_dof</code>	Number of DOF

void setCode (size_t dof, int code)

Assign code `code` to DOF number `dof`.

Parameters

<code>in</code>	<code>dof</code>	index of dof for assignment.
<code>in</code>	<code>code</code>	Value of code to assign.

int getCode (size_t dof = 1) const

Return code for a given DOF of node.

Default value is 1

Node* operator() (size_t i) const

Operator `()`.

Return pointer to node of local label `i`.

size_t getNodeLabel (size_t i) const

Return node label.

Parameters

<code>in</code>	<code>i</code>	Local label of node for which global label is returned
-----------------	----------------	--

int isOnBoundary () const

Say if current edge is a boundary edge or not.

Note this information is available only if boundary edges were determined. See class [Mesh](#)

Node* operator() (size_t i)

Operator ().

Returns pointer to node of local label `i`

7.18 EdgeList Class Reference

Class to construct a list of edges having some common properties.

Public Member Functions

- [EdgeList](#) ([Mesh](#) &ms)
Constructor using a [Mesh](#) instance.
- [~EdgeList](#) ()
Destructor.
- void [selectCode](#) (int code, int dof=1)
Select edges having a given code for a given degree of freedom.
- void [unselectCode](#) (int code, int dof=1)
Unselect edges having a given code for a given degree of freedom.
- size_t [getNbEdges](#) () const
Return number of selected edges.
- void [top](#) ()
Reset list of edges at its top position (Non constant version)
- void [top](#) () const
Reset list of edges at its top position (Constant version)
- [Edge](#) * [get](#) ()
Return pointer to current edge and move to next one (Non constant version)
- [Edge](#) * [get](#) () const
Return pointer to current edge and move to next one (Constant version)

7.18.1 Detailed Description

Class to construct a list of edges having some common properties.

This class enables choosing multiple selection criteria by using function `select...`. However, the intersection of these properties must be empty.

7.18.2 Member Function Documentation

void selectCode (int code, int dof = 1)

Select edges having a given code for a given degree of freedom.

Parameters

<code>in</code>	<code>code</code>	Code that edges share
<code>in</code>	<code>dof</code>	Degree of Freedom label [Default: 1]

void unselectCode (int code, int dof = 1)

Unselect edges having a given code for a given degree of freedom.

Parameters

<code>in</code>	<code>code</code>	Code of edges to exclude
<code>in</code>	<code>dof</code>	Degree of Freedom label [Default: 1]

7.19 Eigen Class Reference

Class to find the smallest eigenvalues and corresponding eigenvectors in a generalized eigenproblem using the Bathe subspace iteration method.

Public Member Functions

- [Eigen](#) ()
Default constructor. Sets default parameters.
- [Eigen](#) ([SkSMatrix](#)< double > &K, [SkSMatrix](#)< double > &M)
Constructor for Symmetric Skyline Matrices.
- [Eigen](#) ([SkSMatrix](#)< double > &K, [Vect](#)< double > &M)
Constructor for Symmetric Skyline Matrices.
- [Eigen](#) ([DSMatrix](#)< double > &K)
Constructor for a dense matrix.
- [~Eigen](#) ()
Destructor.
- void [setMatrix](#) ([SkSMatrix](#)< double > &K, [SkSMatrix](#)< double > &M)
Set matrix instances (Symmetric matrices).
- void [setMatrix](#) ([SkSMatrix](#)< double > &K, [Vect](#)< double > &M)
Set matrix instances (Symmetric matrices).
- void [setMatrix](#) ([DSMatrix](#)< double > &K)
Set matrix instance (Symmetric matrix).
- int [runSubSpace](#) (int nb_eigv, int ss_dim=0)
Run the subspace iteration solver.
- void [setSubspaceDimension](#) (int dim)
Define the subspace dimension.
- void [setMaxIter](#) (int max_it)
set maximal number of iterations.
- int [getNbIter](#) () const
Return actual number of performed iterations.
- void [setTolerance](#) (double eps)
set tolerance value
- int [checkSturm](#) (int &nb_found, int &nb_lost)
Check how many eigenvalues have been found using Sturm sequence method.

- double [getEigenValue](#) (int n) const
Return the n-th eigenvalue.
- void [getEigenVector](#) (int n, [Vect](#)< double > &v) const
Return the n-th eigenvector.

7.19.1 Detailed Description

Class to find the smallest eigenvalues and corresponding eigenvectors in a generalized eigenproblem using the Bathe subspace iteration method.

7.19.2 Constructor & Destructor Documentation

Eigen (SkSMatrix< double > & K, SkSMatrix< double > & M)

Constructor for Symmetric Skyline Matrices.

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	"Mass" matrix

Note

The generalized eigenvalue problem is defined by $Kx = \lambda Mx$, where *K* and *M* are referred to as stiffness and mass matrix. The mass matrix can be diagonal (lumped) or not (consistent). In the first case, it is better to choose a constructor where this matrix is stored in a vector.

Eigen (SkSMatrix< double > & K, Vect< double > & M)

Constructor for Symmetric Skyline Matrices.

Parameters

in	<i>K</i>	"Stiffness" matrix
in	<i>M</i>	Diagonal "Mass" matrix stored as a Vect instance

Note

The generalized eigenvalue problem is defined by $Kx = \lambda Mx$, where *K* and *M* are referred to as stiffness and mass matrix.

Eigen (DSMatrix< double > & K)

Constructor for a dense matrix.

Parameters

in	<i>K</i>	Matrix for which eigenmodes are sought.
--------------------	----------	---

7.19.3 Member Function Documentation

void setMatrix (SkSMatrix< double > & K, SkSMatrix< double > & M)

Set matrix instances (Symmetric matrices).

This function is to be used when the default constructor is applied. Case where the mass matrix is consistent.

Parameters

in	K	Stiffness matrix instance
in	M	Mass matrix instance

void setMatrix (SkSMatrix< double > & K, Vect< double > & M)

Set matrix instances (Symmetric matrices).

This function is to be used when the default constructor is applied. Case where the mass matrix is (lumped) diagonal and stored in a vector.

Parameters

in	K	Stiffness matrix instance
in	M	Mass matrix instance where diagonal terms are stored as a vector.

void setMatrix (DSMatrix< double > & K)

Set matrix instance (Symmetric matrix).

This function is to be used when the default constructor is applied. Case of a standard (not generalized) eigen problem is to be solved

Parameters

in	K	Stiffness matrix instance
-----------	-----	---------------------------

int runSubSpace (int nb_eigv, int ss_dim = 0)

Run the subspace iteration solver.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to be extracted
in	<i>ss_dim</i>	Subspace dimension. Must be at least equal to the number eigenvalues to seek. [Default: nb_eigv

Returns

1: Normal execution. Convergence has been achieved. 2: Convergence for eigenvalues has not been attained.

void setSubspaceDimension (int dim)

Define the subspace dimension.

Parameters

in	<i>dim</i>	Subspace dimension. Must be larger or equal to the number of wanted eigenvalues. By default this value will be set to the number of wanted eigenvalues
-----------	------------	--

void setTolerance (double eps)

set tolerance value

Parameters

in	<i>eps</i>	Convergence tolerance for eigenvalues [Default: 1.e-8]
----	------------	--

int checkSturm (int & nb_found, int & nb_lost)

Check how many eigenvalues have been found using Sturm sequence method.

Parameters

out	<i>nb_found</i>	number of eigenvalues actually found
out	<i>nb_lost</i>	number of eigenvalues missing

Returns

- 0, Successful completion of subroutine.
- 1, No convergent eigenvalues found.

void getEigenVector (int n, Vect< double > & v) const

Return the n-th eigenvector.

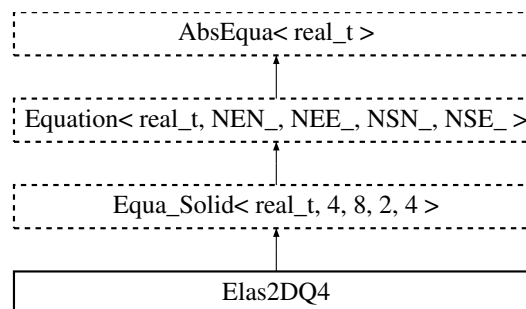
Parameters

in	<i>n</i>	Label of eigenvector (They are stored in ascending order of eigenvalues)
in	<i>v</i>	Vect instance where the eigenvector is stored.

7.20 Elas2DQ4 Class Reference

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.

Inheritance diagram for Elas2DQ4:



Public Member Functions

- [Elas2DQ4](#) ()
Default Constructor.
- [Elas2DQ4](#) (const [Element](#) *el)
Constructor using element data.
- [Elas2DQ4](#) (const [Side](#) *sd)
Constructor using side data.
- [Elas2DQ4](#) (const [Element](#) *element, const [Vect](#)< [real.t](#) > &u, const [real.t](#) &time=0.)

- Constructor using element and previous time data.*

 - `Elas2DQ4` (const `Side` *side, const `Vect`< `real_t` > &u, const `real_t` &time=0.)

Constructor using side and previous time data.

 - `~Elas2DQ4` ()

Destructor.

 - void `PlaneStrain` ()

Set plane strain hypothesis.

 - void `PlaneStrain` (`real_t` E, `real_t` nu)

Set plane strain hypothesis by giving values of Young's modulus and Poisson ratio.

 - void `PlaneStress` ()

Set plane stress hypothesis.

 - void `PlaneStress` (`real_t` E, `real_t` nu)

Set plane stress hypothesis by giving values of Young's modulus and Poisson ratio.

 - void `LMassToLHS` (`real_t` coef=1.)

Add element lumped mass contribution to matrix after multiplication by coef [Default: 1].

 - void `LMassToRHS` (`real_t` coef=1.)

Add element lumped mass contribution to right-hand side after multiplication by coef [Default: 1].

 - void `LMass` (`real_t` coef=1.)

Add element lumped mass contribution to matrix and right-hand side after multiplication by coef [Default: 1].

 - void `Mass` (`real_t` coef=1.)

Add element consistent mass contribution to matrix and right-hand side after multiplication by coef [Default: 1].

 - void `Deviator` (`real_t` coef=1.)

Add element deviatoric matrix to left-hand side after multiplication by coef [Default: 1].

 - void `DeviatorToRHS` (`real_t` coef=1.)

Add element deviatoric contribution to right-hand side after multiplication by coef [Default: 1].

 - void `Dilatation` (`real_t` coef=1.)

Add element dilatational contribution to left-hand side after multiplication by coef [Default: 1].

 - void `DilatationToRHS` (`real_t` coef=1.)

Add element dilatational contribution to right hand side after multiplication by coef [Default: 1].

 - void `BodyRHS` (`UserData`< `real_t` > &ud)

Add body right-hand side term to right hand side after multiplication by coef

 - void `BodyRHS` (const `Vect`< `real_t` > &bf, int opt=`GLOBAL_ARRAY`)

Add body right-hand side term to right hand side.

 - void `BoundaryRHS` (`UserData`< `real_t` > &ud)

Add boundary right-hand side term to right hand side after multiplication by coef

 - void `BoundaryRHS` (const `Vect`< `real_t` > &sf)

Add boundary right-hand side term to right hand side.

 - int `SignoriniContact` (`UserData`< `real_t` > &ud, `real_t` coef=1.e07)

Penalty Signorini contact side contribution to matrix and right-hand side.

 - void `Strain` (`LocalVect`< `real_t`, 3 > &eps)

Calculate strains at element barycenter.

 - void `Stress` (`LocalVect`< `real_t`, 3 > &s, `real_t` &vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

 - void `Stress` (`LocalVect`< `real_t`, 3 > &sigma, `LocalVect`< `real_t`, 3 > &s, `real_t` &vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

- virtual void **MassToLHS** (**real_t** coef=1)
Add consistent mass contribution to left-hand side.
- virtual void **MassToRHS** (**real_t** coef=1)
Add consistent mass contribution to right-hand side.
- void **setLumpedMass** ()
Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.
- void **setMass** ()
Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.
- virtual void **Stiffness** (**real_t** coef=1)
Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void **StiffnessToRHS** (**real_t** coef=1)
Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- void **setDilatation** ()
Add dilatation matrix to left and/or right-hand side taking into account time.
- void **setDeviator** ()
Add deviator matrix to left and/or right-hand side taking into account time integration scheme.
- void **setStiffness** ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- void **buildEigen** (**SkSMatrix**< **real_t** > &K, **SkSMatrix**< **real_t** > &M)
Build global stiffness and mass matrices for the eigen system.
- void **buildEigen** (**SkSMatrix**< **real_t** > &K, **Vect**< **real_t** > &M)
Build global stiffness and mass matrices for the eigen system.
- void **updateBC** (const **Element** &el, const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **updateBC** (const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **DiagBC** (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void **LocalNodeVector** (**Vect**< **real_t** > &b)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void **ElementNodeVectorSingleDOF** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementSideVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementVector** (const **Vect**< **real_t** > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void **SideVector** (const **Vect**< **real_t** > &b)
Localize Side Vector.
- void **ElementNodeCoordinates** ()

- Localize coordinates of element nodes.*
- void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*
- void `ElementAssembly` (`Matrix`< `real_t` > *A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`TrMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`Vect`< `real_t` > &v)
- Assemble element vector into global one.*
- void `DGElementAssembly` (`Matrix`< `real_t` > *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`TrMatrix`< `real_t` > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `SideAssembly` (`Matrix`< `real_t` > *A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`Vect`< `real_t` > &v)
- Assemble side (edge or face) vector into global one.*
- void `AxbAssembly` (const `Element` &el, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
- Assemble product of element matrix by element vector into global vector.*
- void `AxbAssembly` (const `Side` &sd, const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
- Assemble product of side matrix by side vector into global vector.*
- size_t `getNbNodes` () const
- Return number of element nodes.*
- size_t `getNbEq` () const
- Return number of element equations.*
- `real_t` * `A` ()
- Return element matrix as a C-array.*
- `real_t` * `sA` ()

- Return side matrix as a C-array.*

 - `real_t * b ()`
- Return element right-hand side as a C-array.*

 - `real_t * sb ()`
- Return side right-hand side as a C-array.*

 - `real_t * Prev ()`
- Return element matrix as a C-array.*

 - `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
- Return element matrix as a LocalMatrix instance.*

 - `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
- Return side matrix as a LocalMatrix instance.*

 - `LocalVect< real_t, NEE_ > & Eb ()`
- Return element right-hand side as a LocalVect instance.*

 - `LocalVect< real_t, NEE_ > & Ep ()`
- Return element matrix as a C-array.*

 - `void setInitialSolution (const Vect< real_t > &u)`
- Set initial solution (previous time step)*

 - `real_t setMaterialProperty (const string &exp, const string &prop)`
- Define a material property by an algebraic expression.*

 - `void setMesh (class Mesh &m)`
- Define mesh and renumber DOFs after removing imposed ones.*

 - `Mesh & getMesh () const`
- Return reference to Mesh instance.*

 - `LinearSolver< real_t > & getLinearSolver ()`
- Return reference to linear solver instance.*

 - `void setSolver (int ls, int pc=IDENT_PREC)`
- Choose solver for the linear system.*

 - `int solveEigenProblem (int nb_eigv, bool g=false)`
- Compute eigenvalues and eigenvectors.*

 - `real_t getEigenValue (int n) const`
- Return the n-th eigenvalue.*

 - `void getEigenvector (int n, Vect< real_t > &v) const`
- Store the eigenvector corresponding to a given eigenvalue.*

 - `class Eigen & getEigenSolver ()`
- Return reference to eigenproblem solver.*

Protected Member Functions

- `void Young (const real_t &E)`

Set (constant) Young modulus.
- `void Young (const string &exp)`

Set Young modulus given by an algebraic expression.
- `void Poisson (const real_t &nu)`

Set (constant) Poisson ratio.
- `void Poisson (const string &exp)`

Set Poisson ratio given by an algebraic expression.
- `void Density (const real_t &rho)`

- *Set (constant) density.*
• void [Density](#) (const string &exp)
Set density given by an algebraic expression.
- void [setMaterial](#) ()
Set material properties.
- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.20.1 Detailed Description

To build element equations for 2-D linearized elasticity using 4-node quadrilaterals.

This class enables building finite element arrays for linearized isotropic elasticity problem in 2-D domains using 4-Node quadrilaterals.

Unilateral contact is handled using a penalty function. Note that members calculating element arrays have as an argument a real coef that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.20.2 Constructor & Destructor Documentation

Elas2DQ4 ()

Default Constructor.

Constructs an empty equation.

Elas2DQ4 (const [Element](#) * element, const [Vect](#)< real_t > & u, const real_t & time = 0.)

Constructor using element and previous time data.

Parameters

in	<i>element</i>	Pointer to element
in	<i>u</i>	Vect instance containing solution at previous time step
in	<i>time</i>	Current time value [Default: 0]

Elas2DQ4 (const [Side](#) * side, const [Vect](#)< real_t > & u, const real_t & time = 0.)

Constructor using side and previous time data.

Parameters

in	<i>side</i>	Pointer to side
in	<i>u</i>	Vect instance containing solution at previous time step
in	<i>time</i>	Current time value [Default: 0]

7.20.3 Member Function Documentation

void PlaneStrain (real_t E, real_t nu)

Set plane strain hypothesis by giving values of Young's modulus and Poisson ratio.

Parameters

<code>in</code>	E	Young's modulus
<code>in</code>	ν	Poisson ratio

void PlaneStress (`real_t` E , `real_t` ν)

Set plane stress hypothesis by giving values of Young's modulus and Poisson ratio.

Parameters

<code>in</code>	E	Young's modulus
<code>in</code>	ν	Poisson ratio

void DilatationToRHS (`real_t` $\text{coef} = 1.$) [virtual]

Add element dilatational contribution to right hand side after multiplication by coef [Default: 1].

To use for explicit formulations

Reimplemented from [Equa_Solid< real_t, 4, 8, 2, 4 >](#).

void BodyRHS (`UserData< real_t >` & ud)

Add body right-hand side term to right hand side after multiplication by coef

Body forces are deduced from [UserData](#) instance ud

void BodyRHS (`const Vect< real_t >` & bf , `int` $\text{opt} = \text{GLOBAL_ARRAY}$)

Add body right-hand side term to right hand side.

Parameters

<code>in</code>	bf	Vector containing source at element nodes (DOF by DOF).
<code>in</code>	opt	Vector is local (LOCAL_ARRAY) with size 8 or global (GLOBAL_ARRAY) with size = Total number of DOF [Default: GLOBAL_ARRAY].

void BoundaryRHS (`UserData< real_t >` & ud)

Add boundary right-hand side term to right hand side after multiplication by coef

Boundary forces are deduced from [UserData](#) instance ud

void BoundaryRHS (`const Vect< real_t >` & sf)

Add boundary right-hand side term to right hand side.

Parameters

<code>in</code>	sf	Vector containing source at element nodes (DOF by DOF).
-----------------	-------------	---

Warning

The vector sf is sidewise constant, *i.e.* its size is twice the number of sides.

int SignoriniContact (`UserData< real_t >` & ud , `real_t` $\text{coef} = 1.e07$)

Penalty Signorini contact side contribution to matrix and right-hand side.

Parameters

in	<i>ud</i>	UserData instance defining contact data
in	<i>coef</i>	Penalty value by which the added term is multiplied [Default: 1.e07]

Returns

0 if no contact was realized on this side, 1 otherwise

void Strain (LocalVect< real_t, 3 > & eps)

Calculate strains at element barycenter.

Parameters

out	<i>eps</i>	Vector containing strains in element
-----	------------	--------------------------------------

void Stress (LocalVect< real_t, 3 > & s, real_t & vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

Parameters

out	<i>s</i>	LocalVect containing principal stresses in element
out	<i>vm</i>	Value of Von-Mises stress in element

void Stress (LocalVect< real_t, 3 > & sigma, LocalVect< real_t, 3 > & s, real_t & vm)

Calculate principal stresses and Von-Mises stress at element barycenter.

Parameters

out	<i>sigma</i>	Vector containing principal stresses in element
out	<i>s</i>	Vector containing principal stresses in element
out	<i>vm</i>	Value of Von-Mises stress in element

virtual void MassToLHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
----	-------------	--

virtual void MassToRHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
----	-------------	--

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	K	Stiffness matrix
in	M	Vector containing diagonal mass matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
----	------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
-----------	-----------------	--

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer `_theElement`

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
----	----------	--

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an <code>SkSMatrix</code> instance
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an <code>SkMatrix</code> instance
-----------	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	sd	Reference to Side instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
-----------	-----------	---

in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
-----------	-----------	--

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real.t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

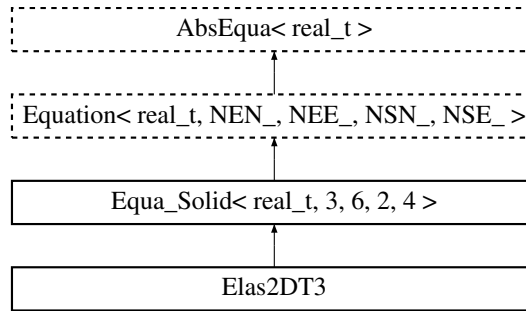
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.21 Elas2DT3 Class Reference

To build element equations for 2-D linearized elasticity using 3-node triangles.

Inheritance diagram for Elas2DT3:



Public Member Functions

- `Elas2DT3 ()`
Default Constructor.
- `Elas2DT3 (const Element *el)`
Constructor using element data.
- `Elas2DT3 (const Side *sd)`
Constructor using side data.
- `Elas2DT3 (const Element *el, const Vect< real_t > &u, real_t time=0.)`
Constructor using element, previous time solution u and time value.
- `Elas2DT3 (const Element *el, const Vect< real_t > &u, real_t time, real_t deltat, int scheme)`
Constructor for an element (transient case) with specification of time integration scheme.
- `Elas2DT3 (const Side *sd, const Vect< real_t > &u, real_t time=0.)`
Constructor using side, previous time solution u and time value.
- `Elas2DT3 (const Side *sd, const Vect< real_t > &u, real_t time, real_t deltat, int scheme)`
Constructor for a side (transient case) with specification of time integration scheme.
- `~Elas2DT3 ()`
Destructor.
- `void Media (real_t E, real_t nu, real_t rho)`
Set media properties.
- `void PlaneStrain ()`
Set plane strain hypothesis.
- `void PlaneStrain (real_t E, real_t nu)`
Set plane strain hypothesis by giving values of Young's modulus E and Poisson ratio ν
- `void PlaneStress ()`
Set plane stress hypothesis.
- `void PlaneStress (real_t E, real_t nu)`
Set plane stress hypothesis by giving values of Young's modulus E and Poisson ratio ν
- `void LMassToLHS (real_t coef=1.)`
Add element lumped mass contribution to matrix after multiplication by $coef$
- `void LMassToRHS (real_t coef=1.)`
Add element lumped mass contribution to right-hand side after multiplication by $coef$
- `void LMass (real_t coef=1.)`
Add element lumped mass contribution to matrix and right-hand side after multiplication by $coef$
- `void MassToLHS (real_t coef=1.)`
Add element consistent mass contribution to matrix after multiplication by $coef$
- `void MassToRHS (real_t coef=1.)`

- Add element consistent mass contribution to right-hand side after multiplication by coef*

 - void `Mass` (`real_t` coef=1.)
- Add element consistent mass contribution to matrix and right-hand side after multiplication by coef*

 - void `Deviator` (`real_t` coef=1.)
- Add element deviatoric matrix to left-hand side after multiplication by coef*

 - void `DeviatorToRHS` (`real_t` coef=1.)
- Add element deviatoric contribution to right-hand side after multiplication by coef*

 - void `Dilatation` (`real_t` coef=1.)
- Add element dilatational contribution to left-hand side after multiplication by coef*

 - void `DilatationToRHS` (`real_t` coef=1.)
- Add element dilatational contribution to right-hand side after multiplication by coef*

 - void `BodyRHS` (`UserData`< `real_t` > &ud)
- Add body right-hand side term to right-hand side after multiplication by coef*

 - void `BodyRHS` (const `Vect`< `real_t` > &f, int opt=`GLOBAL_ARRAY`)
- Add body right-hand side term to right hand side.*

 - void `BoundaryRHS` (`UserData`< `real_t` > &ud)
- Add boundary right-hand side term to right hand side after multiplication by coef*

 - void `BoundaryRHS` (const `Vect`< `real_t` > &f)
- Add boundary right-hand side term to right hand side.*

 - int `SignoriniContact` (`UserData`< `real_t` > &ud, `real_t` coef=1.e07)
- Penalty Signorini contact side contribution to matrix and right-hand side.*

 - int `SignoriniContact` (`Vect`< `real_t` > &f, `real_t` coef=1.e07)
- Penalty Signorini contact side contribution to matrix and right-hand side.*

 - void `Reaction` (`Vect`< `real_t` > &r)
- Calculate reactions.*

 - void `ContactPressure` (const `Vect`< `real_t` > &f, `real_t` penal, `Point`< `real_t` > &p)
- Calculate contact pressure.*

 - void `Strain` (`Vect`< `real_t` > &eps)
- Calculate strains in element.*

 - void `Stress` (`Vect`< `real_t` > &s, `real_t` &vm)
- Calculate principal stresses and Von-Mises stress in element.*

 - void `Periodic` (`real_t` coef=1.e20)
- Add contribution of periodic boundary condition (by a penalty technique).*

 - void `setLumpedMass` ()
- Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.*

 - void `setMass` ()
- Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.*

 - virtual void `Stiffness` (`real_t` coef=1)
- Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void `StiffnessToRHS` (`real_t` coef=1)
- Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - void `setDilatation` ()
- Add dilatation matrix to left and/or right-hand side taking into account time.*

- void `setDeviator ()`
Add deviator matrix to left and/or right-hand side taking into account time integration scheme.
- void `setStiffness ()`
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- void `buildEigen (SkSMatrix< real_t > &K, SkSMatrix< real_t > &M)`
Build global stiffness and mass matrices for the eigen system.
- void `buildEigen (SkSMatrix< real_t > &K, Vect< real_t > &M)`
Build global stiffness and mass matrices for the eigen system.
- void `updateBC (const Element &el, const Vect< real_t > &bc)`
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC (const Vect< real_t > &bc)`
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC (int dof_type=NODE_DOF, int dof=0)`
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector (Vect< real_t > &b)`
Localize Element Vector from a Vect instance.
- void `ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEE_ > &be)`
Localize Element Vector from a Vect instance.
- void `ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be, int dof)`
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be)`
Localize Element Vector from a Vect instance.
- void `ElementSideVector (const Vect< real_t > &b, LocalVect< real_t, NSE_ > &be)`
Localize Element Vector from a Vect instance.
- void `ElementVector (const Vect< real_t > &b, int dof_type=NODE_FIELD, int flag=0)`
Localize Element Vector.
- void `SideVector (const Vect< real_t > &b)`
Localize Side Vector.
- void `ElementNodeCoordinates ()`
Localize coordinates of element nodes.
- void `SideNodeCoordinates ()`
Localize coordinates of side nodes.
- void `ElementAssembly (Matrix< real_t > *A)`
Assemble element matrix into global one.
- void `ElementAssembly (SkSMatrix< real_t > &A)`
Assemble element matrix into global one.
- void `ElementAssembly (SkMatrix< real_t > &A)`
Assemble element matrix into global one.
- void `ElementAssembly (SpMatrix< real_t > &A)`
Assemble element matrix into global one.
- void `ElementAssembly (TrMatrix< real_t > &A)`
Assemble element matrix into global one.
- void `ElementAssembly (Vect< real_t > &v)`
Assemble element vector into global one.
- void `DGElementAssembly (Matrix< real_t > *A)`
Assemble element matrix into global one for the Discontinuous Galerkin approximation.

- void `DGElementAssembly (SkMatrix< real_t > &A)`
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (SkMatrix< real_t > &A)`
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (SpMatrix< real_t > &A)`
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (TrMatrix< real_t > &A)`
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly (Matrix< real_t > *A)`
Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SkMatrix< real_t > &A)`
Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SkMatrix< real_t > &A)`
Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SpMatrix< real_t > &A)`
Assemble side (edge or face) matrix into global one.
- void `SideAssembly (Vect< real_t > &v)`
Assemble side (edge or face) vector into global one.
- void `AxbAssembly (const Element &el, const Vect< real_t > &x, Vect< real_t > &b)`
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly (const Side &sd, const Vect< real_t > &x, Vect< real_t > &b)`
Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes ()` const
Return number of element nodes.
- size_t `getNbEq ()` const
Return number of element equations.
- `real_t * A ()`
Return element matrix as a C-array.
- `real_t * sA ()`
Return side matrix as a C-array.
- `real_t * b ()`
Return element right-hand side as a C-array.
- `real_t * sb ()`
Return side right-hand side as a C-array.
- `real_t * Prev ()`
Return element matrix as a C-array.
- `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
Return element matrix as a LocalMatrix instance.
- `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
Return side matrix as a LocalMatrix instance.
- `LocalVect< real_t, NEE_ > & Eb ()`
Return element right-hand side as a LocalVect instance.
- `LocalVect< real_t, NEE_ > & Ep ()`
Return element matrix as a C-array.
- void `setInitialSolution (const Vect< real_t > &u)`
Set initial solution (previous time step)

- `real_t setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.
- `LinearSolver< real_t >` & `getLinearSolver` ()
Return reference to linear solver instance.
- void `setSolver` (int ls, int pc=IDENT_PREC)
Choose solver for the linear system.
- int `solveEigenProblem` (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue` (int n) const
Return the n-th eigenvalue.
- void `getEigenvector` (int n, `Vect< real_t >` &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class `Eigen` & `getEigenSolver` ()
Return reference to eigenproblem solver.

Protected Member Functions

- void `Young` (const `real_t` &E)
Set (constant) Young modulus.
- void `Young` (const string &exp)
Set Young modulus given by an algebraic expression.
- void `Poisson` (const `real_t` &nu)
Set (constant) Poisson ratio.
- void `Poisson` (const string &exp)
Set Poisson ratio given by an algebraic expression.
- void `Density` (const `real_t` &rho)
Set (constant) density.
- void `Density` (const string &exp)
Set density given by an algebraic expression.
- void `setMaterial` ()
Set material properties.
- void `Init` (const `Element` *el)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.21.1 Detailed Description

To build element equations for 2-D linearized elasticity using 3-node triangles.

This class enables building finite element arrays for linearized isotropic elasticity problem in 2-D domains using 3-Node triangles.

Unilateral contact is handled using a penalty function. Note that members calculating element arrays have as an argument a real coef that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.21.2 Constructor & Destructor Documentation

Elas2DT3 ()

Default Constructor.

Constructs an empty equation.

Elas2DT3 (const Element * el, const Vect< real_t > & u, real_t time = 0.)

Constructor using element, previous time solution u and time value.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].

Elas2DT3 (const Element * el, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for an element (transient case) with specification of time integration scheme.

Parameters

in	<i>el</i>	Pointer to element.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value.
in	<i>deltat</i>	Time step.
in	<i>scheme</i>	Time Integration Scheme: To be chosen among the enumerated values: <ul style="list-style-type: none"> • FORWARD_EULER: Forward Euler scheme • BACKWARD_EULER: Backward Euler scheme, • CRANK_NICOLSON: Crank-Nicolson Euler scheme.

Elas2DT3 (const Side * sd, const Vect< real_t > & u, real_t time = 0.)

Constructor using side, previous time solution u and time value.

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].

Elas2DT3 (const Side * sd, const Vect< real_t > & u, real_t time, real_t deltat, int scheme)

Constructor for a side (transient case) with specification of time integration scheme.

Parameters

in	<i>sd</i>	Pointer to side.
in	<i>u</i>	Vect instance that contains solution at previous time step.
in	<i>time</i>	Current time value [Default: 0].
in	<i>deltat</i>	Time step.
in	<i>scheme</i>	Time Integration Scheme

7.21.3 Member Function Documentation

void Media (real_t E, real_t nu, real_t rho)

Set media properties.

Useful to override material properties deduced from mesh file.

void LMassToLHS (real_t coef = 1.) [virtual]

Add element lumped mass contribution to matrix after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void LMassToRHS (real_t coef = 1.) [virtual]

Add element lumped mass contribution to right-hand side after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void LMass (real_t coef = 1.) [virtual]

Add element lumped mass contribution to matrix and right-hand side after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void MassToLHS (real_t coef = 1.) [virtual]

Add element consistent mass contribution to matrix after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void MassToRHS (real_t coef = 1.) [virtual]

Add element consistent mass contribution to right-hand side after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void Mass (real_t coef = 1.) [virtual]

Add element consistent mass contribution to matrix and right-hand side after multiplication by coef

Parameters

in	coef	Coefficient to multiply by added term [Default: 1].
----	------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void Deviator (real_t coef = 1.) [virtual]

Add element deviatoric matrix to left-hand side after multiplication by coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void DeviatorToRHS (real_t coef = 1.) [virtual]

Add element deviatoric contribution to right-hand side after multiplication by coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void Dilatation (real_t coef = 1.) [virtual]

Add element dilatational contribution to left-hand side after multiplication by coef

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1].
-----------	-------------	---

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void DilatationToRHS (real_t coef = 1.) [virtual]

Add element dilatational contribution to right-hand side after multiplication by coef

To use for explicit formulations

Parameters

in	<i>coef</i>	Coefficient to multiply by added term [Default: 1]
-----------	-------------	--

Reimplemented from [Equa_Solid< real_t, 3, 6, 2, 4 >](#).

void BodyRHS (UserData< real_t > & ud)

Add body right-hand side term to right-hand side after multiplication by coef

Body forces are deduced from [UserData](#) instance ud

void BodyRHS (const Vect< real_t > & f, int opt = GLOBAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>f</i>	Vector containing source at element nodes (DOF by DOF)
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 6 or global (GLOBAL_ARRAY) with size = Number of element DOF [Default: GLOBAL_ARRAY].

void BoundaryRHS (UserData< real_t > & ud)

Add boundary right-hand side term to right hand side after multiplication by coef

Parameters

in	<i>ud</i>	UserData instance defining boundary forces
-----------	-----------	--

void BoundaryRHS (const Vect< real_t > & f)

Add boundary right-hand side term to right hand side.

Parameters

in	f	Vect instance that contains constant traction to impose to side.
-----------	----------	--

int SignoriniContact (UserData< real_t > & ud, real_t coef = 1.e07)

Penalty Signorini contact side contribution to matrix and right-hand side.

Parameters

in	ud	UserData instance defining contact data
in	coef	Penalty value by which the added term is multiplied [Default: 1.e07]

Returns

= 0 if no contact is achieved on this side, 1 otherwise

int SignoriniContact (Vect< real_t > & f, real_t coef = 1.e07)

Penalty Signorini contact side contribution to matrix and right-hand side.

Parameters

in	f	Vect instance that contains contact data
in	coef	Penalty value by which the added term is multiplied [Default: 1.e07]

Returns

= 0 if no contact is achieved on this side, 1 otherwise

void Reaction (Vect< real_t > & r)

Calculate reactions.

This function can be invoked in postprocessing

Parameters

in	r	Reaction on the side
-----------	----------	----------------------

void ContactPressure (const Vect< real_t > & f, real_t penal, Point< real_t > & p)

Calculate contact pressure.

This function can be invoked in postprocessing

Parameters

in	f	
in	penal	Penalty parameter that was used to impose contact condition
out	p	Contact pressure

void Strain (Vect< real_t > & eps)

Calculate strains in element.

This function can be invoked in postprocessing.

void Stress (Vect< real_t > & s, real_t & vm)

Calculate principal stresses and Von-Mises stress in element.

Parameters

in	<i>s</i>	vector of principal stresses
in	<i>vm</i>	Von-Mises stress. This function can be invoked in postprocessing.

void Periodic (real_t coef = 1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC_A on one side and PERIODIC_B on the opposite side.

Parameters

in	<i>coef</i>	Value of penalty parameter [Default: 1.e20]
-----------	-------------	---

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Vector containing diagonal mass matrix

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMMatrix</code> instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

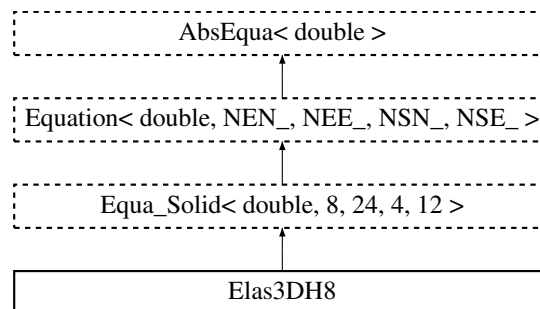
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.22 Elas3DH8 Class Reference

To build element equations for 3-D linearized elasticity using 8-node hexahedra.

Inheritance diagram for Elas3DH8:



Public Member Functions

- [Elas3DH8](#) ()
Default Constructor.
- [Elas3DH8](#) (const [Element](#) *el)
Constructor using element data.
- [Elas3DH8](#) (const [Side](#) *sd)
Constructor using side data.
- [Elas3DH8](#) (const [Element](#) *element, const [Vect](#)< double > &u, const double &time=0.)
Constructor using element, previous time solution u and time value.
- [Elas3DH8](#) (const [Side](#) *side, const [Vect](#)< double > &u, const double &time=0.)
Constructor using side, previous time solution u and time value.
- [~Elas3DH8](#) ()
Destructor.
- void [LMassToLHS](#) (double coef=1.)
Add element lumped mass contribution to matrix after multiplication by coef.
- void [LMassToRHS](#) (double coef=1.)
Add element lumped mass contribution to right-hand side after multiplication by coef.
- void [LMass](#) (double coef)
Add element lumped mass contribution to right-hand and left-hand sides after multiplication by coef.
- void [Mass](#) (double coef=1.)
Add element lumped mass contribution to matrix and right-hand side after multiplication by coef.
- void [Deviator](#) (double coef=1.)
Add element deviatoric matrix to left-hand side after multiplication by coef.
- void [DeviatorToRHS](#) (double coef=1.)
Add element deviatoric matrix to right-hand side after multiplication by coef.
- void [Dilatation](#) (double coef=1.)

- Add element dilatational contribution to left hand-side after multiplication by coef.*

 - void [DilatationToRHS](#) (double coef=1.)
- Add element dilatational contribution to right hand-side after multiplication by coef.*

 - void [BodyRHS](#) ([UserData](#)< double > &ud)
- Add body right-hand side term to right hand side.*

 - void [BoundaryRHS](#) (const [Vect](#)< double > &f)
- Add boundary right-hand side term to right hand side.*

 - void [BodyRHS](#) (const [Vect](#)< double > &bf, int opt=[LOCAL_ARRAY](#))
- Add body right-hand side term to right hand side.*

 - virtual void [MassToLHS](#) ([real_t](#) coef=1)
- Add consistent mass contribution to left-hand side.*

 - virtual void [MassToRHS](#) ([real_t](#) coef=1)
- Add consistent mass contribution to right-hand side.*

 - void [setLumpedMass](#) ()
- Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.*

 - void [setMass](#) ()
- Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.*

 - virtual void [Stiffness](#) ([real_t](#) coef=1)
- Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [StiffnessToRHS](#) ([real_t](#) coef=1)
- Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - void [setDilatation](#) ()
- Add dilatation matrix to left and/or right-hand side taking into account time.*

 - void [setDeviator](#) ()
- Add deviator matrix to left and/or right-hand side taking into account time integration scheme.*

 - void [setStiffness](#) ()
- Add convection contribution to left and/or right-hand side taking into account time integration scheme.*

 - void [buildEigen](#) ([SkSMatrix](#)< double > &K, [SkSMatrix](#)< double > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void [buildEigen](#) ([SkSMatrix](#)< double > &K, [Vect](#)< double > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void [updateBC](#) (const [Element](#) &el, const [Vect](#)< double > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [updateBC](#) (const [Vect](#)< double > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [DiagBC](#) (int dof_type=[NODE_DOF](#), int dof=0)
- Update element matrix to impose bc by diagonalization technique.*

 - void [LocalNodeVector](#) ([Vect](#)< double > &b)
- Localize Element Vector from a Vect instance.*

 - void [ElementNodeVector](#) (const [Vect](#)< double > &b, [LocalVect](#)< double, [NEE_](#) > &be)
- Localize Element Vector from a Vect instance.*

 - void [ElementNodeVector](#) (const [Vect](#)< double > &b, [LocalVect](#)< double, [NEN_](#) > &be, int dof)
- Localize Element Vector from a Vect instance.*

- void `ElementNodeVectorSingleDOF` (const `Vect< double > &b`, `LocalVect< double, NEN_ > &be`)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect< double > &b`, `LocalVect< double, NSE_ > &be`)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect< double > &b`, int dof_type=NODE.FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect< double > &b`)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix< double > *A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< double > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< double > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< double > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< double > &A`)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< double > &v`)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< double > *A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< double > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< double > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< double > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix< double > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix< double > *A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< double > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< double > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix< double > &A`)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect< double > &v`)
Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element &el`, const `Vect< double > &x`, `Vect< double > &b`)
Assemble product of element matrix by element vector into global vector.

- void `AxbAssembly` (const `Side` &sd, const `Vect`< double > &x, `Vect`< double > &b)
Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes` () const
Return number of element nodes.
- size_t `getNbEq` () const
Return number of element equations.
- double * `A` ()
Return element matrix as a C-array.
- double * `sA` ()
Return side matrix as a C-array.
- double * `b` ()
Return element right-hand side as a C-array.
- double * `sb` ()
Return side right-hand side as a C-array.
- double * `Prev` ()
Return element matrix as a C-array.
- `LocalMatrix`< double, NEE_, NEE_ > & `EA` ()
Return element matrix as a LocalMatrix instance.
- `LocalMatrix`< double, NSE_, NSE_ > & `SA` ()
Return side matrix as a LocalMatrix instance.
- `LocalVect`< double, NEE_ > & `Eb` ()
Return element right-hand side as a LocalVect instance.
- `LocalVect`< double, NEE_ > & `Ep` ()
Return element matrix as a C-array.
- void `setInitialSolution` (const `Vect`< double > &u)
Set initial solution (previous time step)
- real_t `setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.
- `LinearSolver`< double > & `getLinearSolver` ()
Return reference to linear solver instance.
- void `setSolver` (int ls, int pc=IDENT_PREC)
Choose solver for the linear system.
- int `solveEigenProblem` (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- real_t `getEigenValue` (int n) const
Return the n-th eigenvalue.
- void `getEigenvector` (int n, `Vect`< real_t > &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class `Eigen` & `getEigenSolver` ()
Return reference to eigenproblem solver.

Protected Member Functions

- void **Young** (const **real_t** &E)
Set (constant) Young modulus.
- void **Young** (const string &exp)
Set Young modulus given by an algebraic expression.
- void **Poisson** (const **real_t** &nu)
Set (constant) Poisson ratio.
- void **Poisson** (const string &exp)
Set Poisson ratio given by an algebraic expression.
- void **Density** (const **real_t** &rho)
Set (constant) density.
- void **Density** (const string &exp)
Set density given by an algebraic expression.
- void **setMaterial** ()
Set material properties.
- void **Init** (const **Element** *el)
Set element arrays to zero.
- void **Init** (const **Side** *sd)
Set side arrays to zero.

7.22.1 Detailed Description

To build element equations for 3-D linearized elasticity using 8-node hexahedra.

This class enables building finite element arrays for linearized isotropic elasticity problem in 3-D domains using 8-Node hexahedra.

Note that members calculating element arrays have as an argument a double **coef** that is multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.22.2 Constructor & Destructor Documentation

Elas3DH8 ()

Default Constructor.

Constructs an empty equation.

7.22.3 Member Function Documentation

void BodyRHS (**UserData**< **double** > & **ud**)

Add body right-hand side term to right hand side.

Body forces are deduced from **UserData** instance *ud*.

void BoundaryRHS (**const Vect**< **double** > & **f**)

Add boundary right-hand side term to right hand side.

Parameters

in	f	Vector containing traction (boundary force) at sides
-----------	----------	--

void BodyRHS (**const Vect**< **double** > & **bf**, **int opt** = **LOCAL_ARRAY**)

Add body right-hand side term to right hand side.

Parameters

in	<i>bf</i>	Vector containing source at element nodes (DOF by DOF).
in	<i>opt</i>	Vector is local (<i>LOCAL_ARRAY</i>) with size 24 or global (<i>GLOBAL_ARRAY</i>) with size = Number of element DOF.

virtual void MassToLHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
-----------	-------------	--

virtual void MassToRHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

void buildEigen (SkSMatrix< double > & K, SkSMatrix< double > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void buildEigen (SkSMatrix< double > & K, Vect< double > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Vector containing diagonal mass matrix

void updateBC (const Element & el, const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<i>in</i>	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [Default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides
<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <i>dof</i> is handled in the system

void LocalNodeVector (Vect< double > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <i>ePrev</i> . This member function is to be used if a constructor with <i>Element</i> was invoked.
-----------	----------	---

void ElementNodeVector (const Vect< double > & b, LocalVect< double , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< double > & b, LocalVect< double , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< double > & b, LocalVect< double , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< double > & b, LocalVect< double , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< double > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides

<i>in</i>	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system The resulting local vector can be accessed by attribute <i>ePrev</i> .
-----------	-------------	---

Remarks

This member function is to be used if a constructor with *Element* was invoked. It uses the *Element* pointer *_theElement*

void SideVector (const Vect< double > & b) [inherited]

Localize Side Vector.
Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides The resulting local vector can be accessed by attribute <i>ePrev</i> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with *Side* was invoked. It uses the *Side* pointer *_theSide*

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array *_x[0]*, *_x[1]*, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the *Side* pointer *_theSide*

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array *_x[0]*, *_x[1]*, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the *Element* pointer *_theElement*

void ElementAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< double > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< double > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< double > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	sd	Reference to Side instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector (Vect instance)

real.t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	exp	Algebraic expression
in	prop	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

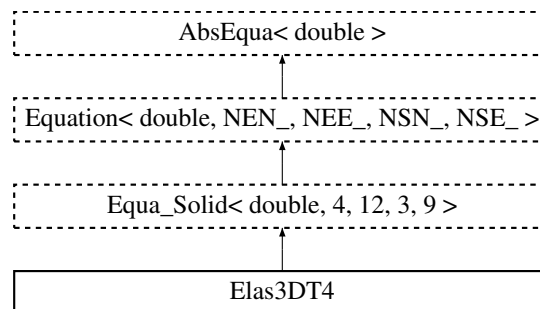
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.23 Elas3DT4 Class Reference

To build element equations for 3-D linearized elasticity using 4-node tetrahedra.

Inheritance diagram for Elas3DT4:



Public Member Functions

- **Elas3DT4** ()
Default Constructor.
- **Elas3DT4** (const **Element** *el)
Constructor using element data.
- **Elas3DT4** (const **Side** *sd)
Constructor using side data.
- **Elas3DT4** (const **Element** *element, const **Vect**< double > &u, const double &time=0.)
Constructor using element and previous time data.
- **Elas3DT4** (const **Side** *side, const **Vect**< double > &u, const double &time=0.)
Constructor using side and previous time data.
- **~Elas3DT4** ()
Destructor.
- void **Media** (double E, double nu, double rho)
Set Media properties.
- void **LMassToLHS** (double coef=1)
Add element lumped mass contribution to matrix after multiplication by coef.
- void **LMassToRHS** (double coef=1)
Add element lumped mass contribution to right-hand side after multiplication by coef.
- void **LMass** (double coef)
Add element lumped mass contribution to matrix and right-hand side after multiplication by coef.
- void **Deviator** (double coef=1.)
Add element deviatoric matrix to left hand-side after multiplication by coef.
- void **DeviatorToRHS** (double coef=1.)
Add element deviatoric matrix to right hand-side after multiplication by coef.
- void **Dilatation** (double coef=1.)

- Add element dilatational contribution to left-hand side after multiplication by coef.*

 - void [DilatationToRHS](#) (double coef=1.)
- Add element dilatational contribution to right-hand side after multiplication by coef.*

 - void [BodyRHS](#) ([UserData](#)< double > &ud)
- Add body right-hand side term to right hand side after multiplication by coef.*

 - void [BodyRHS](#) (const [Vect](#)< double > &f, int opt=[LOCAL_ARRAY](#))
- Add body right-hand side term to right hand side.*

 - void [BoundaryRHS](#) (const [Vect](#)< double > &f)
- Add boundary right-hand side term to right hand side.*

 - void [buildEigen](#) ([SkSMMatrix](#)< double > &K, [Vect](#)< double > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - virtual void [MassToLHS](#) ([real_t](#) coef=1)
- Add consistent mass contribution to left-hand side.*

 - virtual void [MassToRHS](#) ([real_t](#) coef=1)
- Add consistent mass contribution to right-hand side.*

 - void [setLumpedMass](#) ()
- Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.*

 - void [setMass](#) ()
- Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.*

 - virtual void [Mass](#) ([real_t](#) coef=1)
- Add consistent mass matrix to left-hand side after multiplication by coef [Default: 1].*

 - virtual void [Stiffness](#) ([real_t](#) coef=1)
- Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [StiffnessToRHS](#) ([real_t](#) coef=1)
- Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - void [setDilatation](#) ()
- Add dilatation matrix to left and/or right-hand side taking into account time.*

 - void [setDeviator](#) ()
- Add deviator matrix to left and/or right-hand side taking into account time integration scheme.*

 - void [setStiffness](#) ()
- Add convection contribution to left and/or right-hand side taking into account time integration scheme.*

 - void [buildEigen](#) ([SkSMMatrix](#)< double > &K, [SkSMMatrix](#)< double > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void [updateBC](#) (const [Element](#) &el, const [Vect](#)< double > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [updateBC](#) (const [Vect](#)< double > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [DiagBC](#) (int dof_type=[NODE_DOF](#), int dof=0)
- Update element matrix to impose bc by diagonalization technique.*

 - void [LocalNodeVector](#) ([Vect](#)< double > &b)
- Localize Element Vector from a Vect instance.*

 - void [ElementNodeVector](#) (const [Vect](#)< double > &b, [LocalVect](#)< double, [NEE_](#) > &be)
- Localize Element Vector from a Vect instance.*

- void `ElementNodeVector` (const `Vect`< double > &b, `LocalVect`< double, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect`< double > &b, `LocalVect`< double, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect`< double > &b, `LocalVect`< double, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect`< double > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect`< double > &b)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix`< double > *A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix`< double > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix`< double > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix`< double > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix`< double > &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect`< double > &v)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix`< double > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix`< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix`< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix`< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix`< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix`< double > *A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix`< double > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix`< double > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix`< double > &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect`< double > &v)
Assemble side (edge or face) vector into global one.

- void `AxbAssembly` (const `Element` &el, const `Vect`< double > &x, `Vect`< double > &b)
Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect`< double > &x, `Vect`< double > &b)
Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes` () const
Return number of element nodes.
- size_t `getNbEq` () const
Return number of element equations.
- double * `A` ()
Return element matrix as a C-array.
- double * `sA` ()
Return side matrix as a C-array.
- double * `b` ()
Return element right-hand side as a C-array.
- double * `sb` ()
Return side right-hand side as a C-array.
- double * `Prev` ()
Return element matrix as a C-array.
- `LocalMatrix`< double, NEE_, NEE_ > & `EA` ()
Return element matrix as a LocalMatrix instance.
- `LocalMatrix`< double, NSE_, NSE_ > & `SA` ()
Return side matrix as a LocalMatrix instance.
- `LocalVect`< double, NEE_ > & `Eb` ()
Return element right-hand side as a LocalVect instance.
- `LocalVect`< double, NEE_ > & `Ep` ()
Return element matrix as a C-array.
- void `setInitialSolution` (const `Vect`< double > &u)
Set initial solution (previous time step)
- real_t `setMaterialProperty` (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void `setMesh` (class `Mesh` &m)
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh` & `getMesh` () const
Return reference to Mesh instance.
- `LinearSolver`< double > & `getLinearSolver` ()
Return reference to linear solver instance.
- void `setSolver` (int ls, int pc=IDENT_PREC)
Choose solver for the linear system.
- int `solveEigenProblem` (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- real_t `getEigenValue` (int n) const
Return the n-th eigenvalue.
- void `getEigenvector` (int n, `Vect`< real_t > &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class `Eigen` & `getEigenSolver` ()
Return reference to eigenproblem solver.

Protected Member Functions

- void **Young** (const **real_t** &E)
Set (constant) Young modulus.
- void **Young** (const string &exp)
Set Young modulus given by an algebraic expression.
- void **Poisson** (const **real_t** &nu)
Set (constant) Poisson ratio.
- void **Poisson** (const string &exp)
Set Poisson ratio given by an algebraic expression.
- void **Density** (const **real_t** &rho)
Set (constant) density.
- void **Density** (const string &exp)
Set density given by an algebraic expression.
- void **setMaterial** ()
Set material properties.
- void **Init** (const **Element** *el)
Set element arrays to zero.
- void **Init** (const **Side** *sd)
Set side arrays to zero.

7.23.1 Detailed Description

To build element equations for 3-D linearized elasticity using 4-node tetrahedra.

This class enables building finite element arrays for linearized isotropic elasticity problem in 3-D domains using 4-Node tetrahedra.

7.23.2 Member Function Documentation

void Media (double E, double nu, double rho)

Set Media properties.

Parameters

in	<i>E</i>	Young's modulus
in	<i>nu</i>	Poisson ratio
in	<i>rho</i>	Density

void BodyRHS (UserData< double > & ud)

Add body right-hand side term to right hand side after multiplication by *coef*.

Body forces are deduced from **UserData** instance *ud*.

void BodyRHS (const Vect< double > & f, int opt = LOCAL_ARRAY)

Add body right-hand side term to right hand side.

Parameters

in	<i>f</i>	Vect instance containing source at element nodes (DOF by DOF).
in	<i>opt</i>	Vector is local (LOCAL_ARRAY) with size 12 or global (GLOBAL_ARRAY) with size = Number of element DOF.

void BoundaryRHS (const Vect< double > & f)

Add boundary right-hand side term to right hand side.

Parameters

in	<i>f</i>	Vect instance that contains constant traction to impose to side.
-----------	----------	--

void buildEigen (SkSMatrix< double > & K, Vect< double > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Vector containing diagonal mass matrix

virtual void MassToLHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
-----------	-------------	--

virtual void MassToRHS (real_t coef = 1) [virtual], [inherited]

Add consistent mass contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

void buildEigen (SkSMatrix< double > & K, SkSMatrix< double > & M) [inherited]

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void updateBC (const Element & el, const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
-----------	-----------	---------------------------------------

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

void updateBC (const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<i>in</i>	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< double > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
-----------	----------	---

void ElementNodeVector (const Vect< double > & b, LocalVect< double , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector


```
void ElementNodeVector ( const Vect< double > & b, LocalVect< double , NEN_ > & be,  
int dof ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< double > & b, LocalVect< double , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< double > & b, LocalVect< double , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< double > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides

<i>in</i>	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system The resulting local vector can be accessed by attribute <i>ePrev</i> .
-----------	-------------	---

Remarks

This member function is to be used if a constructor with *Element* was invoked. It uses the *Element* pointer *_theElement*

void SideVector (const Vect< double > & b) [inherited]

Localize Side Vector.
Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides The resulting local vector can be accessed by attribute <i>ePrev</i> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with *Side* was invoked. It uses the *Side* pointer *_theSide*

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array *_x[0]*, *_x[1]*, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the *Side* pointer *_theSide*

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array *_x[0]*, *_x[1]*, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the *Element* pointer *_theElement*

void ElementAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< double > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void SideAssembly (Matrix< double > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkSMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< double > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

<i>in</i>	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

<i>in</i>	<i>el</i>	Reference to Element instance
<i>in</i>	<i>x</i>	Global vector to multiply by (Vect instance)
<i>out</i>	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

<i>in</i>	<i>sd</i>	Reference to Side instance
<i>in</i>	<i>x</i>	Global vector to multiply by (Vect instance)
<i>out</i>	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

<i>in</i>	<i>exp</i>	Algebraic expression
<i>in</i>	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.24 Element Class Reference

To store and treat finite element geometric information.

Public Member Functions

- [Element](#) ()
Default constructor.
- [Element](#) (size_t label, const string &shape)
Constructor initializing label, shape of element.
- [Element](#) (size_t label, int shape)
Constructor initializing label, shape of element.
- [Element](#) (size_t label, const string &shape, int c)
Constructor initializing label, shape and code of element.
- [Element](#) (size_t label, int shape, int c)
Constructor initializing label, shape and code of element.
- [Element](#) (const [Element](#) &el)
Copy constructor.
- [~Element](#) ()
Destructor.
- void [setLabel](#) (size_t i)
Define label of element.
- void [setCode](#) (int c)
Define code of element.
- void [setCode](#) (const string &exp, int code)
Define code by a boolean algebraic expression invoking coordinates of element nodes.
- void [Add](#) ([Node](#) *node)
Insert a node at end of list of nodes of element.
- void [Add](#) ([Node](#) *node, int n)
Insert a node and set its local node number.
- void [Replace](#) (size_t label, [Node](#) *node)
Replace a node at a given local label.
- void [Replace](#) (size_t label, [Side](#) *side)
Replace a side at a given local label.
- void [Add](#) ([Side](#) *sd)
Assign Side to Element.
- void [Add](#) ([Side](#) *sd, int k)
Assign Side to Element with assigned local label.
- void [Add](#) ([Element](#) *el)
Add a neighbor element.
- void [set](#) ([Element](#) *el, int n)

- Add a neighbor element and set its label.*

 - void `setDOF` (size_t i, size_t dof)

Define label of DOF.
- void `setCode` (size_t dof, int code)

Assign code to a DOF.
- void `setNode` (size_t i, `Node` *node)

Assign a node given by its pointer as the i-th node of element.
- void `setNbDOF` (size_t i)

Set number of degrees of freedom of element.
- void `setFirstDOF` (size_t i)

Set label of first DOF in element.
- int `getShape` () const

Return element shape.
- size_t `getLabel` () const

Return label of element.
- size_t `n` () const

Return label of element.
- int `getCode` () const

Return code of element.
- size_t `getNbNodes` () const

Return number of element nodes.
- size_t `getNbVertices` () const

Return number of element vertices.
- size_t `getNbSides` () const

Return number of element sides (Constant version)
- size_t `getNbEq` () const

Return number of element equations.
- size_t `getNbDOF` () const

return element nb of DOF
- size_t `getDOF` (size_t i=1) const

Return element DOF label.
- size_t `getFirstDOF` () const

Return element first DOF label.
- size_t `getNodeLabel` (size_t n) const

Return global label of node of local label i.
- size_t `getSideLabel` (size_t n) const

Return global label of side of local label i.
- `Node` * `getPtrNode` (size_t i) const

Return pointer to node of label i (Local labelling).
- `Node` * `operator()` (size_t i) const

Operator ().
- `Side` * `getPtrSide` (size_t i) const

Return pointer to side of label i (Local labelling).
- int `Contains` (const `Node` *n) const

Say if element contains given node.
- int `Contains` (const `Side` *s) const

- Say if element contains given side.*

 - `Element * getNeighborElement (size_t i) const`
Return pointer to element Neighboring element.
 - `size_t getNbNeigElements () const`
Return number of neighboring elements.
 - `double getMeasure () const`
Return measure of element.
 - `Point< double > getUnitNormal (size_t i) const`
Return outward unit normal to i-th side of element.
 - `bool isOnBoundary () const`
Say if current element is a boundary element or not.
 - `Node * operator() (size_t i)`
Operator ().
 - `int setSide (size_t n, size_t *nd)`
Initialize information on element sides.
 - `bool isActive () const`
Return true or false whether element is active or not.
 - `int getLevel () const`
Return element level `Element` level decreases when element is refined (starting from 0). If the level is 0, then the element has no father.
 - `void setChild (Element *el)`
Assign element as child of current one and assign current element as father This function is principally used when refining is invoked (e.g. for mesh adaption)
 - `Element * getChild (size_t i) const`
Return pointer to i-th child element Return null pointer is no childs.
 - `size_t getNbChilds () const`
Return number of children of element.
 - `Element * getParent () const`
Return pointer to parent element Return null if no parent.
 - `size_t lsln (const Node *nd)`
Check if a given node belongs to current element.

7.24.1 Detailed Description

To store and treat finite element geometric information.

Class `Element` enables defining an element of a finite element mesh. The element is given in particular by its shape and a list of nodes. Each node can be accessed by the member function `getPtrNode`. Moreover, class `Mesh` can generate for each element its list of sides. The string that defines the element shape must be chosen according to the following list :

Remarks

Once a `Mesh` instance is constructed, one has access for each `Element` of the mesh to pointers to element sides provided the member function `getAllSides` of `Mesh` has been invoked. With this, an element can be tested to see if it is on the boundary, i.e. if it has at least one side on the boundary

7.24.2 Constructor & Destructor Documentation

`Element (size_t label, const string & shape)`

Constructor initializing label, shape of element.

Parameters

<code>in</code>	<i>label</i>	Label to assign to element.
<code>in</code>	<i>shape</i>	Shape of element (See class description).

Element (`size_t label`, `int shape`)

Constructor initializing label, shape of element.

Parameters

<code>in</code>	<i>label</i>	Label to assign to element.
<code>in</code>	<i>shape</i>	Shape of element (See enum <code>ElementShape</code> in Mesh)

Element (`size_t label`, `const string & shape`, `int c`)

Constructor initializing label, shape and code of element.

Parameters

<code>in</code>	<i>label</i>	Label to assign to element.
<code>in</code>	<i>shape</i>	Shape of element (See class description).
<code>in</code>	<i>c</i>	Code to assign to element (useful for media properties).

Element (`size_t label`, `int shape`, `int c`)

Constructor initializing label, shape and code of element.

Parameters

<code>in</code>	<i>label</i>	Label to assign to element.
<code>in</code>	<i>shape</i>	Shape of element (See enum <code>ElementShape</code> in Mesh).
<code>in</code>	<i>c</i>	Code to assign to element (useful for media properties).

7.24.3 Member Function Documentation

void setLabel (`size_t i`)

Define label of element.

Parameters

<code>in</code>	<i>i</i>	Label to assign to element
-----------------	----------	----------------------------

void setCode (`int c`)

Define code of element.

Parameters

<code>in</code>	<i>c</i>	Code to assign to element.
-----------------	----------	----------------------------

void setCode (`const string & exp`, `int code`)

Define code by a boolean algebraic expression invoking coordinates of element nodes.

Parameters

in	<i>exp</i>	Boolean algebraic expression as required by <code>fparser</code>
in	<i>code</i>	Code to assign to node if the algebraic expression is true

void Add (Node * node)

Insert a node at end of list of nodes of element.

Parameters

in	<i>node</i>	Pointer to Node instance.
-----------	-------------	---

void Add (Node * node, int n)

Insert a node and set its local node number.

Parameters

	<i>node</i>	[in] Pointer to Node instance
in	<i>n</i>	Element node number to assign

void Replace (size_t label, Node * node)

Replace a node at a given local label.

Parameters

in	<i>label</i>	Node to replace.
in	<i>node</i>	Pointer to Node instance to copy to current instance.

void Replace (size_t label, Side * side)

Replace a side at a given local label.

Parameters

in	<i>label</i>	Side to replace.
in	<i>side</i>	Pointer to Side instance to copy to current instance.

void Add (Side * sd)

Assign [Side](#) to [Element](#).

Parameters

in	<i>sd</i>	Pointer to Side instance.
-----------	-----------	---

void Add (Side * sd, int k)

Assign [Side](#) to [Element](#) with assigned local label.

Parameters

in	<i>sd</i>	Pointer to Side instance.
-----------	-----------	---

<i>in</i>	<i>k</i>	Local label.
-----------	----------	--------------

void Add (Element * el)

Add a neighbor element.

Parameters

<i>in</i>	<i>el</i>	Pointer to Element instance
-----------	-----------	---

void set (Element * el, int n)

Add a neighbor element and set its label.

Parameters

<i>in</i>	<i>el</i>	Pointer to Element instance
<i>in</i>	<i>n</i>	Neighbor element number to assign

void setDOF (size_t i, size_t dof)

Define label of DOF.

Parameters

<i>in</i>	<i>i</i>	Index of DOF.
<i>in</i>	<i>dof</i>	Label of DOF to assign.

void setCode (size_t dof, int code)

Assign code to a DOF.

Parameters

<i>in</i>	<i>dof</i>	Index of dof for assignment.
<i>in</i>	<i>code</i>	Code to assign.

Node* operator() (size_t i) const

Operator ().

Return pointer to node of local label *i*.

int Contains (const Node * n) const

Say if element contains given node.

Parameters

<i>in</i>	<i>n</i>	Pointer to Node instance.
-----------	----------	---

Returns

Local node label in element. If 0, the element does not contain this node

int Contains (const Side * s) const

Say if element contains given side.

Parameters

<code>in</code>	<code>s</code>	Pointer to Side instance.
-----------------	----------------	---

Returns

Local side label in element. If 0, the element does not contain this side

Element* getNeighborElement (size_t i) const

Return pointer to element Neighboring element.

Parameters

<code>in</code>	<code>i</code>	Index of element to look for.
-----------------	----------------	-------------------------------

Note

This method returns valid information only if the [Mesh](#) member function [Mesh::getElementNeighborElements\(\)](#) has been called before.

size_t getNbNeigElements () const

Return number of neighboring elements.

Note

This method returns valid information only if the [Mesh](#) member function [Mesh::getElementNeighborElements\(\)](#) has been called before.

double getMeasure () const

Return measure of element.

This member function returns length, area or volume of element. In case of quadrilaterals and hexahedrals it returns determinant of Jacobian of mapping between reference and actual element

Point<double> getUnitNormal (size_t i) const

Return outward unit normal to i-th side of element.

Sides are ordered [node_1,node_2], [node_2,node_3], ...

bool isOnBoundary () const

Say if current element is a boundary element or not.

Note

this information is available only if boundary elements were determined i.e. if member function [Mesh::getBoundarySides](#) or [Mesh::getAllSides](#) has been invoked before.

Node* operator() (size_t i)

Operator ().

Return pointer to node of local label i.

int setSide (size_t n, size_t * nd)

Initialize information on element sides.

This function is to be used to initialize loops over sides.

Parameters

in	<i>n</i>	Label of side.
in	<i>nd</i>	Array of pointers to nodes of the side (<i>nd</i> [0], <i>nd</i> [1], ... point to first, second nodes, ...

void setChild (Element * el)

Assign element as child of current one and assign current element as father This function is principally used when refining is invoked (e.g. for mesh adaption)

Parameters

in	<i>el</i>	Pointer to element to assign
----	-----------	------------------------------

size_t lsln (const Node * nd)

Check if a given node belongs to current element.

Parameters

in	<i>nd</i>	Pointer to node to locate
----	-----------	---------------------------

Returns

local label of node if this one is found, 0 otherwise

7.25 ElementList Class Reference

Class to construct a list of elements having some common properties.

Public Member Functions

- [ElementList](#) ([Mesh](#) &ms)
Constructor using a [Mesh](#) instance.
- [~ElementList](#) ()
Destructor.
- void [selectCode](#) (int code)
Select elements having a given code.
- void [unselectCode](#) (int code)
Unselect elements having a given code.
- void [selectLevel](#) (int level)
Select elements having a given level.
- size_t [getNbElements](#) () const
Return number of selected elements.
- void [top](#) ()
Reset list of elements at its top position (Non constant version)
- void [top](#) () const
Reset list of elements at its top position (Constant version)
- [Element](#) * [get](#) ()
Return pointer to current element and move to next one (Non constant version)
- [Element](#) * [get](#) () const
Return pointer to current element and move to next one (Constant version)

7.25.1 Detailed Description

Class to construct a list of elements having some common properties.

This class enables choosing multiple selection criteria by using function `select...`. However, the intersection of these properties must be empty.

7.25.2 Member Function Documentation

void unselectCode (int code)

Unselect elements having a given code.

Parameters

<code>in</code>	<code>code</code>	Code of elements to exclude
-----------------	-------------------	-----------------------------

void selectLevel (int level)

Select elements having a given level.

Parameters

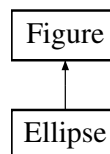
<code>in</code>	<code>level</code>	Level of elements to select
-----------------	--------------------	-----------------------------

Elements having a given level (for mesh adaption) are selected in a list

7.26 Ellipse Class Reference

To store and treat an ellipsoidal figure.

Inheritance diagram for Ellipse:



Public Member Functions

- [Ellipse](#) ()
Default constructor.
- [Ellipse](#) ([Point](#)< double > c, double a, double [b](#), int code=1)
Constructor with given data.
- double [getSignedDistance](#) (const [Point](#)< double > &p) const
Return signed distance of a given point from the current ellipse.
- [Ellipse](#) & [operator+=](#) ([Point](#)< double > a)
Operator +=
- [Ellipse](#) & [operator+=](#) (double a)
*Operator *=*
- void [setCode](#) (int code)
Choose a code for the domain defined by the figure.
- void [getSignedDistance](#) (const [Grid](#) &g, [Vect](#)< double > &d) const
Calculate signed distance to current figure with respect to grid points.

- `double dLine (const Point< double > &p, const Point< double > &a, const Point< double > &b) const`

Compute signed distance from a line.

7.26.1 Detailed Description

To store and treat an ellipsoidal figure.

7.26.2 Constructor & Destructor Documentation

Ellipse ()

Default constructor.

Constructs an ellipse with semimajor axis = 1, and semiminor axis = 1

Ellipse (Point< double > c, double a, double b, int code = 1)

Constructor with given data.

Parameters

in	<i>c</i>	Coordinates of center
in	<i>a</i>	Semimajor axis
in	<i>b</i>	Semiminor axis
in	<i>code</i>	Code to assign to the generated figure [Default: 1]

7.26.3 Member Function Documentation

`double getSignedDistance (const Point< double > & p) const` [virtual]

Return signed distance of a given point from the current ellipse.

The computed distance is negative if p lies in the ellipse, positive if it is outside, and 0 on its boundary

Parameters

in	<i>p</i>	Point<double> instance
----	----------	------------------------

Reimplemented from [Figure](#).

Ellipse& operator+= (Point< double > a)

Operator +=

Translate ellipse by a vector a

Ellipse& operator+= (double a)

Operator *=

Scale ellipse by a factor a

`void getSignedDistance (const Grid & g, Vect< double > & d) const` [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector *d* doesn't need to be sized before invoking this function

double dLine (const [Point](#)< double > & p, const [Point](#)< double > & a, const [Point](#)< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	<i>p</i>	Point for which distance is computed
in	<i>a</i>	First vertex of line
in	<i>b</i>	Second vertex of line

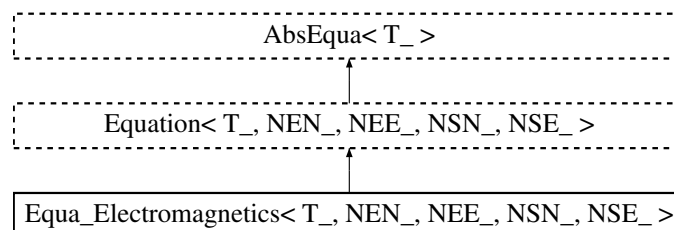
Returns

Signed distance

7.27 Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for Electromagnetics [Equation](#) classes.

Inheritance diagram for Equa_Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< T_ > &b)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be, int dof)
Localize [Element](#) Vector from a [Vect](#) instance.

- void [ElementNodeVectorSingleDOF](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementSideVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NSE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementVector](#) (const [Vect](#)< T_ > &b, int dof_type=NODE.FIELD, int flag=0)
Localize [Element](#) Vector.
- void [SideVector](#) (const [Vect](#)< T_ > &b)
Localize [Side](#) Vector.
- void [ElementNodeCoordinates](#) ()
Localize coordinates of element nodes.
- void [SideNodeCoordinates](#) ()
Localize coordinates of side nodes.
- void [ElementAssembly](#) ([Matrix](#)< T_ > *A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkSMatrix](#)< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkMatrix](#)< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SpMatrix](#)< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([TrMatrix](#)< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([Vect](#)< T_ > &v)
Assemble element vector into global one.
- void [DGElementAssembly](#) ([Matrix](#)< T_ > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkSMatrix](#)< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkMatrix](#)< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SpMatrix](#)< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([TrMatrix](#)< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [SideAssembly](#) ([Matrix](#)< T_ > *A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SkSMatrix](#)< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SkMatrix](#)< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SpMatrix](#)< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([Vect](#)< T_ > &v)
Assemble side (edge or face) vector into global one.
- void [AxbAssembly](#) (const [Element](#) &el, const [Vect](#)< T_ > &x, [Vect](#)< T_ > &b)
Assemble product of element matrix by element vector into global vector.

- void [AxbAssembly](#) (const [Side](#) &sd, const [Vect](#)< T_ > &x, [Vect](#)< T_ > &b)
Assemble product of side matrix by side vector into global vector.
- size_t [getNbNodes](#) () const
Return number of element nodes.
- size_t [getNbEq](#) () const
Return number of element equations.
- T_ * [A](#) ()
Return element matrix as a C-array.
- T_ * [sA](#) ()
Return side matrix as a C-array.
- T_ * [b](#) ()
Return element right-hand side as a C-array.
- T_ * [sb](#) ()
Return side right-hand side as a C-array.
- T_ * [Prev](#) ()
Return element matrix as a C-array.
- [LocalMatrix](#)< T_, NEE_, NEE_ > & [EA](#) ()
Return element matrix as a [LocalMatrix](#) instance.
- [LocalMatrix](#)< T_, NSE_, NSE_ > & [SA](#) ()
Return side matrix as a [LocalMatrix](#) instance.
- [LocalVect](#)< T_, NEE_ > & [Eb](#) ()
Return element right-hand side as a [LocalVect](#) instance.
- [LocalVect](#)< T_, NEE_ > & [Ep](#) ()
Return element matrix as a C-array.
- void [setInitialSolution](#) (const [Vect](#)< T_ > &u)
Set initial solution (previous time step)
- real_t [setMaterialProperty](#) (const string &exp, const string &prop)
Define a material property by an algebraic expression.
- void [setMesh](#) (class [Mesh](#) &m)
Define mesh and renumber DOFs after removing imposed ones.
- [Mesh](#) & [getMesh](#) () const
Return reference to [Mesh](#) instance.
- [LinearSolver](#)< T_ > & [getLinearSolver](#) ()
Return reference to linear solver instance.
- void [setSolver](#) (int ls, int pc=IDENT_PREC)
Choose solver for the linear system.
- int [solveEigenProblem](#) (int nb_eigv, bool g=false)
Compute eigenvalues and eigenvectors.
- real_t [getEigenValue](#) (int n) const
Return the n-th eigenvalue.
- void [getEigenvector](#) (int n, [Vect](#)< real_t > &v) const
Store the eigenvector corresponding to a given eigenvalue.
- class [Eigen](#) & [getEigenSolver](#) ()
Return reference to eigenproblem solver.

Protected Member Functions

- void [MagneticPermeability](#) (const [real_t](#) &mu)
Set (constant) magnetic permeability.
- void [MagneticPermeability](#) (const string &exp)
Set magnetic permeability given by an algebraic expression.
- void [ElectricConductivity](#) (const [real_t](#) &sigma)
Set (constant) electric conductivity.
- void [ElectricConductivity](#) (const string &exp)
set electric conductivity given by an algebraic expression
- void [ElectricResistivity](#) (const [real_t](#) &rho)
Set (constant) electric resistivity.
- void [ElectricResistivity](#) (const string &exp)
Set electric resistivity given by an algebraic expression.
- void [setMaterial](#) ()
Set material properties.
- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.27.1 Detailed Description

**template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equa_-
 Electromagnetics< T_, NEN_, NEE_, NSN_, NSE_ >**

Abstract class for Electromagnetics [Equation](#) classes.

Template Parameters

<T_>	data type (double, float, ...)
<NEN_>	Number of element nodes
<NEE_>	Number of element equations
<NSN_>	Number of side nodes
<NSE_>	Number of side equations

7.27.2 Member Function Documentation

void updateBC (const [Element](#) & el, const [Vect](#)< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const [Vect](#)< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<i>in</i>	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [Default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides
<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <i>dof</i> is handled in the system

void LocalNodeVector (Vect< T_ > & b) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <i>ePrev</i> . This member function is to be used if a constructor with Element was invoked.
-----------	----------	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be)
 [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = NODE_FIELD, int flag = 0)
 [inherited]

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides

<i>in</i>	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.
-----------	-------------	--

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer `_theElement`

void SideVector (const Vect< T_ > & b) [inherited]

Localize [Side](#) Vector.
Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
-----------	----------	--

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one.

Parameters

	<i>A</i>	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

	<i>A</i>	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>in</i>	<i>A</i>	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>in</i>	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>in</i>	<i>A</i>	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< T_ > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void SideAssembly (Matrix< T_ > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< T_ > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	sd	Reference to Side instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	exp	Algebraic expression
in	prop	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

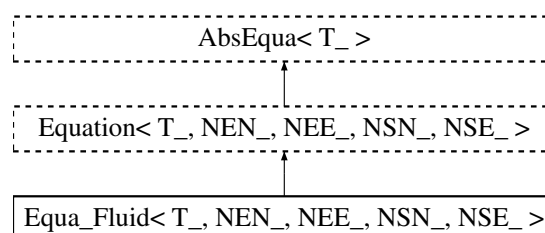
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.28 Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for Fluid Dynamics [Equation](#) classes.

Inheritance diagram for Equa_Fluid< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- [Equa_Fluid](#) ()
Default constructor.
- virtual [~Equa_Fluid](#) ()
Destructor.
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< T_ > &b)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be, int dof)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVectorSingleDOF](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementSideVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NSE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementVector](#) (const [Vect](#)< T_ > &b, int dof_type=NODE_FIELD, int flag=0)
Localize [Element](#) Vector.
- void [SideVector](#) (const [Vect](#)< T_ > &b)
Localize [Side](#) Vector.
- void [ElementNodeCoordinates](#) ()

- Localize coordinates of element nodes.*
- void [SideNodeCoordinates](#) ()
- Localize coordinates of side nodes.*
- void [ElementAssembly](#) (Matrix< T_ > *A)
- Assemble element matrix into global one.*
- void [ElementAssembly](#) (SkSMatrix< T_ > &A)
- Assemble element matrix into global one.*
- void [ElementAssembly](#) (SkMatrix< T_ > &A)
- Assemble element matrix into global one.*
- void [ElementAssembly](#) (SpMatrix< T_ > &A)
- Assemble element matrix into global one.*
- void [ElementAssembly](#) (TrMatrix< T_ > &A)
- Assemble element matrix into global one.*
- void [ElementAssembly](#) (Vect< T_ > &v)
- Assemble element vector into global one.*
- void [DGElementAssembly](#) (Matrix< T_ > *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGElementAssembly](#) (SkSMatrix< T_ > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGElementAssembly](#) (SkMatrix< T_ > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGElementAssembly](#) (SpMatrix< T_ > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGElementAssembly](#) (TrMatrix< T_ > &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [SideAssembly](#) (Matrix< T_ > *A)
- Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) (SkSMatrix< T_ > &A)
- Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) (SkMatrix< T_ > &A)
- Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) (SpMatrix< T_ > &A)
- Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) (Vect< T_ > &v)
- Assemble side (edge or face) vector into global one.*
- void [AxbAssembly](#) (const [Element](#) &el, const Vect< T_ > &x, Vect< T_ > &b)
- Assemble product of element matrix by element vector into global vector.*
- void [AxbAssembly](#) (const [Side](#) &sd, const Vect< T_ > &x, Vect< T_ > &b)
- Assemble product of side matrix by side vector into global vector.*
- size_t [getNbNodes](#) () const
- Return number of element nodes.*
- size_t [getNbEq](#) () const
- Return number of element equations.*
- T_ * [A](#) ()
- Return element matrix as a C-array.*
- T_ * [sA](#) ()

- *Return side matrix as a C-array.*
- `T_ * b ()`
- *Return element right-hand side as a C-array.*
- `T_ * sb ()`
- *Return side right-hand side as a C-array.*
- `T_ * Prev ()`
- *Return element matrix as a C-array.*
- `LocalMatrix< T_, NEE_, NEE_ > & EA ()`
- *Return element matrix as a **LocalMatrix** instance.*
- `LocalMatrix< T_, NSE_, NSE_ > & SA ()`
- *Return side matrix as a **LocalMatrix** instance.*
- `LocalVect< T_, NEE_ > & Eb ()`
- *Return element right-hand side as a **LocalVect** instance.*
- `LocalVect< T_, NEE_ > & Ep ()`
- *Return element matrix as a C-array.*
- `void setInitialSolution (const Vect< T_ > &u)`
- *Set initial solution (previous time step)*
- `real.t setMaterialProperty (const string &exp, const string &prop)`
- *Define a material property by an algebraic expression.*
- `void setMesh (class Mesh &m)`
- *Define mesh and renumber DOFs after removing imposed ones.*
- `Mesh & getMesh () const`
- *Return reference to **Mesh** instance.*
- `LinearSolver< T_ > & getLinearSolver ()`
- *Return reference to linear solver instance.*
- `void setSolver (int ls, int pc=IDENT_PREC)`
- *Choose solver for the linear system.*
- `int solveEigenProblem (int nb_eigv, bool g=false)`
- *Compute eigenvalues and eigenvectors.*
- `real.t getEigenValue (int n) const`
- *Return the n-th eigenvalue.*
- `void getEigenvector (int n, Vect< real.t > &v) const`
- *Store the eigenvector corresponding to a given eigenvalue.*
- `class Eigen & getEigenSolver ()`
- *Return reference to eigenproblem solver.*

Protected Member Functions

- `void Viscosity (const real.t &visc)`
- *Set (constant) Viscosity.*
- `void Viscosity (const string &exp)`
- *Set viscosity given by an algebraic expression.*
- `void Density (const real.t &dens)`
- *Set (constant) Viscosity.*
- `void Density (const string &exp)`
- *Set Density given by an algebraic expression.*
- `void ThermalExpansion (const real.t *e)`

- void `ThermalExpansion` (const string &exp)
Set (constant) thermal expansion coefficient.
- void `setMaterial` ()
Set thermal expansion coefficient given by an algebraic expression.
- void `Init` (const `Element` *el)
Set material properties.
- void `Init` (const `Side` *sd)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.28.1 Detailed Description

template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equa_-Fluid< T_, NEN_, NEE_, NSN_, NSE_ >

Abstract class for Fluid Dynamics `Equation` classes.
Template Parameters

<T_>	data type (double, float, ...)
<NEN_>	Number of element nodes
<NEE_>	Number of element equations
<NSN_>	Number of side nodes
<NSE_>	Number of side equations

7.28.2 Constructor & Destructor Documentation

Equa_Fluid ()

Default constructor.
Constructs an empty equation.

7.28.3 Member Function Documentation

void updateBC (const Element & el, const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.
Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.
Parameters

in	bc	Vector that contains imposed values at all DOFs
----	----	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< T_ > & b) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
----	---	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer *_theElement*

void SideVector (const Vect< T_ > & b) [inherited]

Localize [Side](#) Vector.

Parameters

in	b	<p>Global vector to be localized</p> <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides <p>The resulting local vector can be accessed by attribute <code>ePrev</code>.</p>
----	---	--

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< T_ > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< T_ > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< T_ > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)

out	<i>b</i>	Global vector to add (Vect instance)
-----	----------	---

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

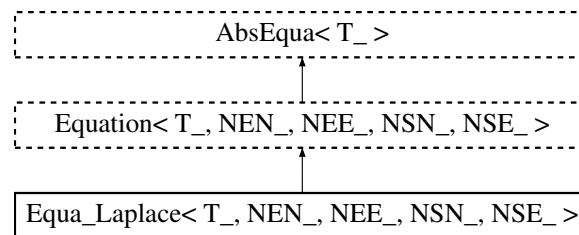
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.29 Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for classes about the Laplace equation.

Inheritance diagram for Equa_Laplace< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- [Equa_Laplace](#) ()
Default constructor.
- virtual [~Equa_Laplace](#) ()
Destructor.
- int [run](#) ()
Solve the equation.
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof.type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< T_ > &b)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be, int dof)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVectorSingleDOF](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEN_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementSideVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NSE_ > &be)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementVector](#) (const [Vect](#)< T_ > &b, int dof.type=NODE_FIELD, int flag=0)
Localize [Element](#) Vector.
- void [SideVector](#) (const [Vect](#)< T_ > &b)

- Localize *Side* Vector.
- void `ElementNodeCoordinates ()`
 Localize coordinates of element nodes.
- void `SideNodeCoordinates ()`
 Localize coordinates of side nodes.
- void `ElementAssembly (Matrix< T_ > *A)`
 Assemble element matrix into global one.
- void `ElementAssembly (SkSMatrix< T_ > &A)`
 Assemble element matrix into global one.
- void `ElementAssembly (SkMatrix< T_ > &A)`
 Assemble element matrix into global one.
- void `ElementAssembly (SpMatrix< T_ > &A)`
 Assemble element matrix into global one.
- void `ElementAssembly (TrMatrix< T_ > &A)`
 Assemble element matrix into global one.
- void `ElementAssembly (Vect< T_ > &v)`
 Assemble element vector into global one.
- void `DGElementAssembly (Matrix< T_ > *A)`
 Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (SkSMatrix< T_ > &A)`
 Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (SkMatrix< T_ > &A)`
 Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (SpMatrix< T_ > &A)`
 Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly (TrMatrix< T_ > &A)`
 Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly (Matrix< T_ > *A)`
 Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SkSMatrix< T_ > &A)`
 Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SkMatrix< T_ > &A)`
 Assemble side (edge or face) matrix into global one.
- void `SideAssembly (SpMatrix< T_ > &A)`
 Assemble side (edge or face) matrix into global one.
- void `SideAssembly (Vect< T_ > &v)`
 Assemble side (edge or face) vector into global one.
- void `AxbAssembly (const Element &el, const Vect< T_ > &x, Vect< T_ > &b)`
 Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly (const Side &sd, const Vect< T_ > &x, Vect< T_ > &b)`
 Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes ()` const
 Return number of element nodes.
- size_t `getNbEq ()` const
 Return number of element equations.
- `T_ * A ()`

- *Return element matrix as a C-array.*
- `T_ * sA ()`
- *Return side matrix as a C-array.*
- `T_ * b ()`
- *Return element right-hand side as a C-array.*
- `T_ * sb ()`
- *Return side right-hand side as a C-array.*
- `T_ * Prev ()`
- *Return element matrix as a C-array.*
- `LocalMatrix< T_, NEE_, NEE_ > & EA ()`
- *Return element matrix as a [LocalMatrix](#) instance.*
- `LocalMatrix< T_, NSE_, NSE_ > & SA ()`
- *Return side matrix as a [LocalMatrix](#) instance.*
- `LocalVect< T_, NEE_ > & Eb ()`
- *Return element right-hand side as a [LocalVect](#) instance.*
- `LocalVect< T_, NEE_ > & Ep ()`
- *Return element matrix as a C-array.*
- `void setInitialSolution (const Vect< T_ > &u)`
- *Set initial solution (previous time step)*
- `real_t setMaterialProperty (const string &exp, const string &prop)`
- *Define a material property by an algebraic expression.*
- `void setMesh (class Mesh &m)`
- *Define mesh and renumber DOFs after removing imposed ones.*
- `Mesh & getMesh () const`
- *Return reference to [Mesh](#) instance.*
- `LinearSolver< T_ > & getLinearSolver ()`
- *Return reference to linear solver instance.*
- `void setSolver (int ls, int pc=IDENT_PREC)`
- *Choose solver for the linear system.*
- `int solveEigenProblem (int nb_eigv, bool g=false)`
- *Compute eigenvalues and eigenvectors.*
- `real_t getEigenValue (int n) const`
- *Return the n-th eigenvalue.*
- `void getEigenvector (int n, Vect< real_t > &v) const`
- *Store the eigenvector corresponding to a given eigenvalue.*
- `class Eigen & getEigenSolver ()`
- *Return reference to eigenproblem solver.*

Protected Member Functions

- `void Init (const Element *el)`
- *Set element arrays to zero.*
- `void Init (const Side *sd)`
- *Set side arrays to zero.*

7.29.1 Detailed Description

```
template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equa_-
Laplace< T_, NEN_, NEE_, NSN_, NSE_ >
```

Abstract class for classes about the Laplace equation.

Template Arguments:

- **T_** : data type (double, float, ...)
- **NEN_** : Number of element nodes
- **NEE_** : Number of element equations
- **NSN_** : Number of side nodes
- **NSE_** : Number of side equations

7.29.2 Constructor & Destructor Documentation

Equa_Laplace ()

Default constructor.

Constructs an empty equation.

7.29.3 Member Function Documentation

void updateBC (const Element & el, const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>el</i>	Reference to current element instance
<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by *_theElement*

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< T_ > & b) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
----	---	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer *_theElement*

void SideVector (const Vect< T_ > & b) [inherited]

Localize [Side](#) Vector.

Parameters

in	b	<p>Global vector to be localized</p> <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides <p>The resulting local vector can be accessed by attribute <code>ePrev</code>.</p>
----	---	--

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void ElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void ElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void ElementAssembly (Vect< T_ > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable theElement

void DGElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< T_ > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< T_ > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)

out	<i>b</i>	Global vector to add (Vect instance)
-----	----------	---

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

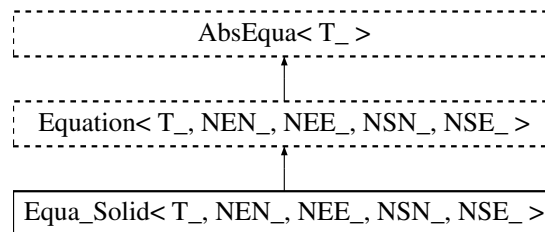
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.30 Equa_Solid< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for Solid Mechanics Finite Element classes.

Inheritance diagram for Equa_Solid< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- [Equa_Solid](#) ()
Default constructor.
- virtual [~Equa_Solid](#) ()
Destructor.
- virtual void [LMassToLHS](#) ([real_t](#) coef=1)
Add lumped mass contribution to left-hand side.
- virtual void [LMassToRHS](#) ([real_t](#) coef=1)
Add lumped mass contribution to right-hand side.
- virtual void [MassToLHS](#) ([real_t](#) coef=1)
Add consistent mass contribution to left-hand side.
- virtual void [MassToRHS](#) ([real_t](#) coef=1)
Add consistent mass contribution to right-hand side.
- void [setLumpedMass](#) ()
Add lumped mass contribution to left and right-hand sides taking into account time integration scheme.
- void [setMass](#) ()
Add consistent mass contribution to left and right-hand sides taking into account time integration scheme.
- virtual void [Mass](#) ([real_t](#) coef=1)
Add consistent mass matrix to left-hand side after multiplication by coef [Default: 1].
- virtual void [LMass](#) ([real_t](#) coef=1)
Add lumped mass matrix to left-hand side after multiplication by coef [Default: 1].
- virtual void [Deviator](#) ([real_t](#) coef=1)
Add deviator matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].
- virtual void [Dilatation](#) ([real_t](#) coef=1)

- Add dilatation matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [DilatationToRHS](#) ([real_t](#) coef=1)
- Add dilatation vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [DeviatorToRHS](#) ([real_t](#) coef=1)
- Add deviator vector to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [Stiffness](#) ([real_t](#) coef=1)
- Add stiffness matrix to left-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - virtual void [StiffnessToRHS](#) ([real_t](#) coef=1)
- Add stiffness matrix to right-hand side taking into account time integration scheme, after multiplication by coef [Default: 1].*

 - void [setDilatation](#) ()
- Add dilatation matrix to left and/or right-hand side taking into account time.*

 - void [setDeviator](#) ()
- Add deviator matrix to left and/or right-hand side taking into account time integration scheme.*

 - void [setStiffness](#) ()
- Add convection contribution to left and/or right-hand side taking into account time integration scheme.*

 - void [buildEigen](#) ([SkSMatrix](#)< [T_](#) > &K, [SkSMatrix](#)< [T_](#) > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void [buildEigen](#) ([SkSMatrix](#)< [T_](#) > &K, [Vect](#)< [T_](#) > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void [updateBC](#) (const [Element](#) &el, const [Vect](#)< [T_](#) > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [updateBC](#) (const [Vect](#)< [T_](#) > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
- Update element matrix to impose bc by diagonalization technique.*

 - void [LocalNodeVector](#) ([Vect](#)< [T_](#) > &b)
- Localize [Element](#) Vector from a [Vect](#) instance.*

 - void [ElementNodeVector](#) (const [Vect](#)< [T_](#) > &b, [LocalVect](#)< [T_](#), [NEE_](#) > &be)
- Localize [Element](#) Vector from a [Vect](#) instance.*

 - void [ElementNodeVector](#) (const [Vect](#)< [T_](#) > &b, [LocalVect](#)< [T_](#), [NEN_](#) > &be, int dof)
- Localize [Element](#) Vector from a [Vect](#) instance.*

 - void [ElementNodeVectorSingleDOF](#) (const [Vect](#)< [T_](#) > &b, [LocalVect](#)< [T_](#), [NEN_](#) > &be)
- Localize [Element](#) Vector from a [Vect](#) instance.*

 - void [ElementSideVector](#) (const [Vect](#)< [T_](#) > &b, [LocalVect](#)< [T_](#), [NSE_](#) > &be)
- Localize [Element](#) Vector from a [Vect](#) instance.*

 - void [ElementVector](#) (const [Vect](#)< [T_](#) > &b, int dof_type=NODE_FIELD, int flag=0)
- Localize [Element](#) Vector.*

 - void [SideVector](#) (const [Vect](#)< [T_](#) > &b)
- Localize [Side](#) Vector.*

 - void [ElementNodeCoordinates](#) ()
- Localize coordinates of element nodes.*

 - void [SideNodeCoordinates](#) ()

- Localize coordinates of side nodes.*
- void [ElementAssembly](#) ([Matrix](#)< T_ > *A)
 - Assemble element matrix into global one.*
- void [ElementAssembly](#) ([SkSMatrix](#)< T_ > &A)
 - Assemble element matrix into global one.*
- void [ElementAssembly](#) ([SkMatrix](#)< T_ > &A)
 - Assemble element matrix into global one.*
- void [ElementAssembly](#) ([SpMatrix](#)< T_ > &A)
 - Assemble element matrix into global one.*
- void [ElementAssembly](#) ([TrMatrix](#)< T_ > &A)
 - Assemble element matrix into global one.*
- void [ElementAssembly](#) ([Vect](#)< T_ > &v)
 - Assemble element vector into global one.*
- void [DGEElementAssembly](#) ([Matrix](#)< T_ > *A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGEElementAssembly](#) ([SkSMatrix](#)< T_ > &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGEElementAssembly](#) ([SkMatrix](#)< T_ > &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGEElementAssembly](#) ([SpMatrix](#)< T_ > &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [DGEElementAssembly](#) ([TrMatrix](#)< T_ > &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void [SideAssembly](#) ([Matrix](#)< T_ > *A)
 - Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) ([SkSMatrix](#)< T_ > &A)
 - Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) ([SkMatrix](#)< T_ > &A)
 - Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) ([SpMatrix](#)< T_ > &A)
 - Assemble side (edge or face) matrix into global one.*
- void [SideAssembly](#) ([Vect](#)< T_ > &v)
 - Assemble side (edge or face) vector into global one.*
- void [AxbAssembly](#) (const [Element](#) &el, const [Vect](#)< T_ > &x, [Vect](#)< T_ > &b)
 - Assemble product of element matrix by element vector into global vector.*
- void [AxbAssembly](#) (const [Side](#) &sd, const [Vect](#)< T_ > &x, [Vect](#)< T_ > &b)
 - Assemble product of side matrix by side vector into global vector.*
- size_t [getNbNodes](#) () const
 - Return number of element nodes.*
- size_t [getNbEq](#) () const
 - Return number of element equations.*
- T_ * [A](#) ()
 - Return element matrix as a C-array.*
- T_ * [sA](#) ()
 - Return side matrix as a C-array.*
- T_ * [b](#) ()

- Return element right-hand side as a C-array.*

 - `T_ * sb ()`
- Return side right-hand side as a C-array.*

 - `T_ * Prev ()`
- Return element matrix as a C-array.*

 - `LocalMatrix< T_, NEE_, NEE_ > & EA ()`
- Return element matrix as a `LocalMatrix` instance.*

 - `LocalMatrix< T_, NSE_, NSE_ > & SA ()`
- Return side matrix as a `LocalMatrix` instance.*

 - `LocalVect< T_, NEE_ > & Eb ()`
- Return element right-hand side as a `LocalVect` instance.*

 - `LocalVect< T_, NEE_ > & Ep ()`
- Return element matrix as a C-array.*

 - `void setInitialSolution (const Vect< T_ > &u)`
- Set initial solution (previous time step)*

 - `real_t setMaterialProperty (const string &exp, const string &prop)`
- Define a material property by an algebraic expression.*

 - `void setMesh (class Mesh &m)`
- Define mesh and renumber DOFs after removing imposed ones.*

 - `Mesh & getMesh () const`
- Return reference to `Mesh` instance.*

 - `LinearSolver< T_ > & getLinearSolver ()`
- Return reference to linear solver instance.*

 - `void setSolver (int ls, int pc=IDENT_PREC)`
- Choose solver for the linear system.*

 - `int solveEigenProblem (int nb_eigv, bool g=false)`
- Compute eigenvalues and eigenvectors.*

 - `real_t getEigenValue (int n) const`
- Return the n-th eigenvalue.*

 - `void getEigenvector (int n, Vect< real_t > &v) const`
- Store the eigenvector corresponding to a given eigenvalue.*

 - `class Eigen & getEigenSolver ()`
- Return reference to eigenproblem solver.*

Protected Member Functions

- `void Young (const real_t &E)`

Set (constant) Young modulus.
- `void Poisson (const real_t &nu)`

Set (constant) Poisson ratio.
- `void Density (const real_t &rho)`

Set (constant) density.
- `void Young (const string &exp)`

Set Young modulus given by an algebraic expression.
- `void Poisson (const string &exp)`

Set Poisson ratio given by an algebraic expression.
- `void Density (const string &exp)`

Set density given by an algebraic expression.

- void [setMaterial](#) ()

Set material properties.

- void [Init](#) (const [Element](#) *el)

Set element arrays to zero.

- void [Init](#) (const [Side](#) *sd)

Set side arrays to zero.

7.30.1 Detailed Description

template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equa_Solid< T_, NEN_, NEE_, NSN_, NSE_ >

Abstract class for Solid Mechanics Finite Element classes.

Template Parameters

<code><T_></code>	data type (double, float, ...)
<code><NEN_></code>	Number of element nodes
<code><NEE_></code>	Number of element equations
<code><NSN_></code>	Number of side nodes
<code><NSE_></code>	Number of side equations

7.30.2 Constructor & Destructor Documentation

Equa_Solid ()

Default constructor.

Constructs an empty equation.

7.30.3 Member Function Documentation

virtual void LMassToLHS (real_t coef = 1) [virtual]

Add lumped mass contribution to left-hand side.

Parameters

<code>in</code>	<code>coef</code>	coefficient to multiply by the matrix before adding [Default: 1]
-----------------	-------------------	--

Reimplemented in [Elas2DT3](#), [Elas2DQ4](#), [Beam3DL2](#), [Elas3DH8](#), [Elas3DT4](#), and [Bar2DL2](#).

virtual void LMassToRHS (real_t coef = 1) [virtual]

Add lumped mass contribution to right-hand side.

Parameters

<code>in</code>	<code>coef</code>	coefficient to multiply by the vector before adding [Default: 1]
-----------------	-------------------	--

Reimplemented in [Elas2DT3](#), [Elas2DQ4](#), [Beam3DL2](#), [Elas3DH8](#), [Elas3DT4](#), and [Bar2DL2](#).

virtual void MassToLHS (real_t coef = 1) [virtual]

Add consistent mass contribution to left-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the matrix before adding [Default: 1]
-----------	-------------	--

Reimplemented in [Elas2DT3](#), [Beam3DL2](#), and [Bar2DL2](#).

virtual void MassToRHS (real_t coef = 1) [virtual]

Add consistent mass contribution to right-hand side.

Parameters

in	<i>coef</i>	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

Reimplemented in [Elas2DT3](#), [Beam3DL2](#), and [Bar2DL2](#).

void buildEigen (SkSMatrix< T_ > & K, SkSMatrix< T_ > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Consistent mass matrix

void buildEigen (SkSMatrix< T_ > & K, Vect< T_ > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	<i>K</i>	Stiffness matrix
in	<i>M</i>	Vector containing diagonal mass matrix

void updateBC (const Element & el, const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< T_ > & b) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
----	---	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer *_theElement*

void SideVector (const Vect< T_ > & b) [inherited]

Localize [Side](#) Vector.

Parameters

in	b	Global vector to be localized <ul style="list-style-type: none">• NODE.FIELD, DOFs are supported by nodes [default]• ELEMENT.FIELD, DOFs are supported by elements• SIDE.FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
----	---	--

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.
Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.
Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one.
Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.
Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< T_ > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< T_ > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< T_ > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

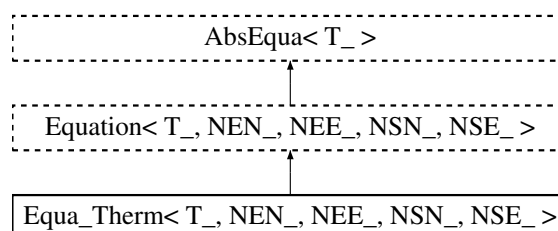
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.31 Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for Heat transfer Finite Element classes.

Inheritance diagram for Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- [Equa_Therm](#) ()
Default constructor.
- virtual [~Equa_Therm](#) ()
Destructor.
- virtual void [setStab](#) ()
Set stabilized formulation.
- virtual void [LCapacityToLHS](#) ([real_t](#) coef=1)
Add lumped capacity contribution to left-hand side.
- virtual void [LCapacityToRHS](#) ([real_t](#) coef=1)
Add lumped capacity contribution to right-hand side.
- virtual void [CapacityToLHS](#) ([real_t](#) coef=1)
Add consistent capacity contribution to left-hand side.
- virtual void [CapacityToRHS](#) ([real_t](#) coef=1)
Add consistent capacity contribution to right-hand side.
- void [setLumpedCapacity](#) ()
Add lumped capacity contribution to left and right-hand sides taking into account time integration scheme.
- void [setCapacity](#) ()
Add consistent capacity contribution to left and right-hand sides taking into account time integration scheme.
- virtual void [Diffusion](#) ([real_t](#) coef=1.)
Add diffusion term to left-hand side.
- virtual void [DiffusionToRHS](#) ([real_t](#) coef=1.)
Add diffusion term to right-hand side.
- void [setDiffusion](#) ()
Add diffusion contribution to left and/or right-hand side taking into account time integration scheme.

- virtual void `Convection` (`real_t` coef=1.)
Add convection term to left-hand side.
- virtual void `ConvectionToRHS` (`real_t` coef=1.)
Add convection term to right-hand side.
- void `setConvection` ()
Add convection contribution to left and/or right-hand side taking into account time integration scheme.
- void `build` ()
Build the linear system of equations.
- int `runTransient` ()
Run one time step.
- int `run` ()
Run the equation.
- void `setRhoCp` (const `real_t` &rhocp)
Set product of Density by Specific heat (constants)
- void `setConductivity` (const `real_t` &diff)
Set (constant) thermal conductivity.
- void `RhoCp` (const string &exp)
Set product of Density by Specific heat given by an algebraic expression.
- void `Conduc` (const string &exp)
Set thermal conductivity given by an algebraic expression.
- void `buildEigen` (`SkSMatrix`< `T_` > &K, `SkSMatrix`< `T_` > &M)
Build global stiffness and mass matrices for the eigen system.
- void `buildEigen` (`SkSMatrix`< `T_` > &K, `Vect`< `T_` > &M)
Build global diffusion and capacity matrices for the eigen system.
- void `updateBC` (const `Element` &el, const `Vect`< `T_` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `updateBC` (const `Vect`< `T_` > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void `LocalNodeVector` (`Vect`< `T_` > &b)
Localize `Element` Vector from a `Vect` instance.
- void `ElementNodeVector` (const `Vect`< `T_` > &b, `LocalVect`< `T_`, `NEE_` > &be)
Localize `Element` Vector from a `Vect` instance.
- void `ElementNodeVector` (const `Vect`< `T_` > &b, `LocalVect`< `T_`, `NEN_` > &be, int dof)
Localize `Element` Vector from a `Vect` instance.
- void `ElementNodeVectorSingleDOF` (const `Vect`< `T_` > &b, `LocalVect`< `T_`, `NEN_` > &be)
Localize `Element` Vector from a `Vect` instance.
- void `ElementSideVector` (const `Vect`< `T_` > &b, `LocalVect`< `T_`, `NSE_` > &be)
Localize `Element` Vector from a `Vect` instance.
- void `ElementVector` (const `Vect`< `T_` > &b, int dof_type=NODE_FIELD, int flag=0)
Localize `Element` Vector.
- void `SideVector` (const `Vect`< `T_` > &b)
Localize `Side` Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.

- void [SideNodeCoordinates](#) ()
Localize coordinates of side nodes.
- void [ElementAssembly](#) (Matrix< T_ > *A)
Assemble element matrix into global one.
- void [ElementAssembly](#) (SkSMatrix< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) (SkMatrix< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) (SpMatrix< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) (TrMatrix< T_ > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) (Vect< T_ > &v)
Assemble element vector into global one.
- void [DGElementAssembly](#) (Matrix< T_ > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) (SkSMatrix< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) (SkMatrix< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) (SpMatrix< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) (TrMatrix< T_ > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [SideAssembly](#) (Matrix< T_ > *A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) (SkSMatrix< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) (SkMatrix< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) (SpMatrix< T_ > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) (Vect< T_ > &v)
Assemble side (edge or face) vector into global one.
- void [AxbAssembly](#) (const [Element](#) &el, const Vect< T_ > &x, Vect< T_ > &b)
Assemble product of element matrix by element vector into global vector.
- void [AxbAssembly](#) (const [Side](#) &sd, const Vect< T_ > &x, Vect< T_ > &b)
Assemble product of side matrix by side vector into global vector.
- size_t [getNbNodes](#) () const
Return number of element nodes.
- size_t [getNbEq](#) () const
Return number of element equations.
- T_ * [A](#) ()
Return element matrix as a C-array.
- T_ * [sA](#) ()
Return side matrix as a C-array.

- `T_ * b ()`
Return element right-hand side as a C-array.
- `T_ * sb ()`
Return side right-hand side as a C-array.
- `T_ * Prev ()`
Return element matrix as a C-array.
- `LocalMatrix< T_, NEE_, NEE_ > & EA ()`
Return element matrix as a `LocalMatrix` instance.
- `LocalMatrix< T_, NSE_, NSE_ > & SA ()`
Return side matrix as a `LocalMatrix` instance.
- `LocalVect< T_, NEE_ > & Eb ()`
Return element right-hand side as a `LocalVect` instance.
- `LocalVect< T_, NEE_ > & Ep ()`
Return element matrix as a C-array.
- `void setInitialSolution (const Vect< T_ > &u)`
Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`
Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`
Return reference to `Mesh` instance.
- `LinearSolver< T_ > & getLinearSolver ()`
Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`
Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n-th eigenvalue.
- `void getEigenVector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void setMaterial ()`
Set material properties.
- `void Init (const Element *el)`
Set element arrays to zero.
- `void Init (const Side *sd)`
Set side arrays to zero.

7.31.1 Detailed Description

`template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equa_Therm< T_, NEN_, NEE_, NSN_, NSE_ >`

Abstract class for Heat transfer Finite Element classes.

Template Parameters

<T_>	data type (real_t, float, ...)
<NEN_>	Number of element nodes
<NEE_>	Number of element equations
<NSN_>	Number of side nodes
<NSE_>	Number of side equations

7.31.2 Constructor & Destructor Documentation

Equa_Therm ()

Default constructor.

Constructs an empty equation.

7.31.3 Member Function Documentation

virtual void setStab () [virtual]

Set stabilized formulation.

Stabilized variational formulations are to be used when the Pclet number is large.

By default, no stabilization is used.

virtual void LCapacityToLHS (real_t coef = 1) [virtual]

Add lumped capacity contribution to left-hand side.

Parameters

in	coef	coefficient to multiply by the matrix before adding [Default: 1]
----	------	--

Reimplemented in [DC2DT3](#), [DC3DT4](#), [DC3DAT3](#), and [DC1DL2](#).

virtual void LCapacityToRHS (real_t coef = 1) [virtual]

Add lumped capacity contribution to right-hand side.

Parameters

in	coef	coefficient to multiply by the vector before adding [Default: 1]
----	------	--

Reimplemented in [DC2DT3](#), [DC3DT4](#), [DC3DAT3](#), and [DC1DL2](#).

virtual void CapacityToLHS (real_t coef = 1) [virtual]

Add consistent capacity contribution to left-hand side.

Parameters

in	coef	coefficient to multiply by the matrix before adding [Default: 1]
----	------	--

Reimplemented in [DC2DT3](#), [DC3DT4](#), [DC3DAT3](#), and [DC1DL2](#).

virtual void CapacityToRHS (real_t coef = 1) [virtual]

Add consistent capacity contribution to right-hand side.

Parameters

in	coef	coefficient to multiply by the vector before adding [Default: 1]
-----------	-------------	--

Reimplemented in [DC2DT3](#), [DC3DT4](#), [DC3DAT3](#), and [DC1DL2](#).

void build ()

Build the linear system of equations.

Before using this function, one must have properly selected appropriate options for:

- The choice of a steady state or transient analysis. By default, the analysis is stationary
- In the case of transient analysis, the choice of a time integration scheme and a lumped or consistent capacity matrix. If transient analysis is chosen, the lumped capacity matrix option is chosen by default, and the implicit Euler scheme is used by default for time integration.

int runTransient ()

Run one time step.

This function performs one time step in equation solving. It is to be used only if a *TRANSIENT* analysis is required.

Returns

Return error from the linear system solver

int run ()

Run the equation.

If the analysis (see function `setAnalysis`) is *STEADY_STATE*, then the function solves the stationary equation.

If the analysis is *TRANSIENT*, then the function performs time stepping until the final time is reached.

void buildEigen (SkSMatrix< T_ > & K, SkSMatrix< T_ > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< T_ > & K, Vect< T_ > & M)

Build global diffusion and capacity matrices for the eigen system.

Case where the capacity matrix is lumped

Parameters

in	K	Diffusion matrix
in	M	Vector containing diagonal capacity matrix

void updateBC (const Element & el, const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const Vect< T_ > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
----	----	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< T_ > & b) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with Element was invoked.
----	---	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)
[inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be) [inherited]

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = *NODE_FIELD*, int flag = 0) [inherited]

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer `_theElement`

void SideVector (const Vect< T_ > & b) [inherited]

Localize [Side](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
----	----------	--

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< T_ > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (Matrix< T_ > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	---	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	---	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< T_ > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< T_ > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< T_ > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
----	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< T_ > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
----	----------	--

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function `getEigen` was called with an argument `nb_eigv` greater or equal to `n`. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

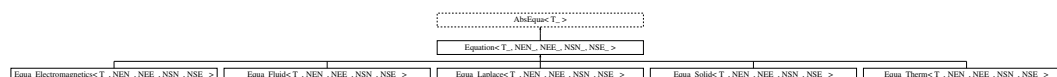
Parameters

in	<i>n</i>	Label of the eigenvalue
out	<i>v</i>	Vect instance containing the corresponding eigenvector. This vector is resized.

7.32 Equation< T_, NEN_, NEE_, NSN_, NSE_ > Class Template Reference

Abstract class for all equation classes.

Inheritance diagram for Equation< T_, NEN_, NEE_, NSN_, NSE_ >:



Public Member Functions

- [Equation](#) ()
Constructor with mesh instance.
- [Equation](#) (class [Mesh](#) &mesh)
Constructor with mesh instance, matrix and right-hand side.
- [Equation](#) (const [Element](#) *el)
Constructor using [Element](#) data.
- [Equation](#) (const [Side](#) *sd)
Constructor using [Side](#) data.
- [Equation](#) (const [Element](#) *el, const [Vect](#)< T_ > &u, const [real_t](#) &time=0)
Constructor using element data, solution at previous time step and time value.
- [Equation](#) (const [Side](#) *sd, const [Vect](#)< T_ > &u, const [real_t](#) &time=0)
Constructor using side data, solution at previous time step and time value.
- virtual [~Equation](#) ()
Destructor.
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< T_ > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< T_ > &b)
Localize [Element](#) Vector from a [Vect](#) instance.
- void [ElementNodeVector](#) (const [Vect](#)< T_ > &b, [LocalVect](#)< T_, NEE_ > &be)

- Localize *Element* Vector from a *Vect* instance.

 - void `ElementNodeVectorSingleDOF` (const `Vect< T_ > &b`, `LocalVect< T_, NEN_ > &be`)
- Localize *Element* Vector from a *Vect* instance.

 - void `ElementNodeVector` (const `Vect< T_ > &b`, `LocalVect< T_, NEN_ > &be`, int dof)
- Localize *Element* Vector from a *Vect* instance.

 - void `ElementSideVector` (const `Vect< T_ > &b`, `LocalVect< T_, NSE_ > &be`)
- Localize *Element* Vector from a *Vect* instance.

 - void `ElementVector` (const `Vect< T_ > &b`, int dof_type=NODE.FIELD, int flag=0)
- Localize *Element* Vector.

 - void `SideVector` (const `Vect< T_ > &b`)
- Localize *Side* Vector.

 - void `ElementNodeCoordinates` ()
- Localize coordinates of element nodes.

 - void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.

 - void `ElementAssembly` (`Matrix< T_ > *A`)
- Assemble element matrix into global one.

 - void `ElementAssembly` (`SkSMatrix< T_ > &A`)
- Assemble element matrix into global one.

 - void `ElementAssembly` (`SkMatrix< T_ > &A`)
- Assemble element matrix into global one.

 - void `ElementAssembly` (`SpMatrix< T_ > &A`)
- Assemble element matrix into global one.

 - void `ElementAssembly` (`TrMatrix< T_ > &A`)
- Assemble element matrix into global one.

 - void `DGElementAssembly` (`Matrix< T_ > *A`)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.

 - void `DGElementAssembly` (`SkSMatrix< T_ > &A`)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.

 - void `DGElementAssembly` (`SkMatrix< T_ > &A`)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.

 - void `DGElementAssembly` (`SpMatrix< T_ > &A`)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.

 - void `DGElementAssembly` (`TrMatrix< T_ > &A`)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.

 - void `SideAssembly` (`Matrix< T_ > *A`)
- Assemble side (edge or face) matrix into global one.

 - void `SideAssembly` (`SkSMatrix< T_ > &A`)
- Assemble side (edge or face) matrix into global one.

 - void `SideAssembly` (`SkMatrix< T_ > &A`)
- Assemble side (edge or face) matrix into global one.

 - void `SideAssembly` (`SpMatrix< T_ > &A`)
- Assemble side (edge or face) matrix into global one.

 - void `ElementAssembly` (`Vect< T_ > &v`)
- Assemble element vector into global one.

 - void `SideAssembly` (`Vect< T_ > &v`)

- Assemble side (edge or face) vector into global one.*
 - void `AxbAssembly` (const `Element` &el, const `Vect`< T_ > &x, `Vect`< T_ > &b)
- Assemble product of element matrix by element vector into global vector.*
 - void `AxbAssembly` (const `Side` &sd, const `Vect`< T_ > &x, `Vect`< T_ > &b)
- Assemble product of side matrix by side vector into global vector.*
 - size_t `getNbNodes` () const
- Return number of element nodes.*
 - size_t `getNbEq` () const
- Return number of element equations.*
 - T_ * `A` ()
- Return element matrix as a C-array.*
 - T_ * `sA` ()
- Return side matrix as a C-array.*
 - T_ * `b` ()
- Return element right-hand side as a C-array.*
 - T_ * `sb` ()
- Return side right-hand side as a C-array.*
 - T_ * `Prev` ()
- Return element matrix as a C-array.*
 - `LocalMatrix`< T_, NEE_, NEE_ > & `EA` ()
- Return element matrix as a LocalMatrix instance.*
 - `LocalMatrix`< T_, NSE_, NSE_ > & `SA` ()
- Return side matrix as a LocalMatrix instance.*
 - `LocalVect`< T_, NEE_ > & `Eb` ()
- Return element right-hand side as a LocalVect instance.*
 - `LocalVect`< T_, NEE_ > & `Ep` ()
- Return element matrix as a C-array.*
 - void `setInitialSolution` (const `Vect`< T_ > &u)
- Set initial solution (previous time step)*
 - real_t `setMaterialProperty` (const string &exp, const string &prop)
- Define a material property by an algebraic expression.*
 - void `setMesh` (class `Mesh` &m)
- Define mesh and renumber DOFs after removing imposed ones.*
 - `Mesh` & `getMesh` () const
- Return reference to Mesh instance.*
 - `LinearSolver`< T_ > & `getLinearSolver` ()
- Return reference to linear solver instance.*
 - void `setSolver` (int ls, int pc=IDENT_PREC)
- Choose solver for the linear system.*
 - int `solveEigenProblem` (int nb_eigv, bool g=false)
- Compute eigenvalues and eigenvectors.*
 - real_t `getEigenValue` (int n) const
- Return the n-th eigenvalue.*
 - void `getEigenVector` (int n, `Vect`< real_t > &v) const
- Store the eigenvector corresponding to a given eigenvalue.*
 - class `Eigen` & `getEigenSolver` ()
- Return reference to eigenproblem solver.*

Protected Member Functions

- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.32.1 Detailed Description

template<class T_, size_t NEN_, size_t NEE_, size_t NSN_, size_t NSE_>class OFELI::Equation< T_, NEN_, NEE_, NSN_, NSE_ >

Abstract class for all equation classes.

Template Arguments:

- **T_** : data type (real_t, float, ...)
- **NEN_** : Number of element nodes
- **NEE_** : Number of element equations
- **NSN_** : Number of side nodes
- **NSN_** : Number of side equations

7.32.2 Constructor & Destructor Documentation

Equation ()

Default constructor. Constructs an "empty" equation

Equation (class Mesh & mesh)

Constructor with mesh instance.

Parameters

in	<i>mesh</i>	Mesh instance
--------------------	-------------	-------------------------------

Equation (class Mesh & mesh, Vect< T_ > & b, real_t & t, real_t & ts)

Constructor with mesh instance, matrix and right-hand side.

Parameters

in	<i>mesh</i>	Mesh instance
in	<i>b</i>	Vect instance containing Right-hand side.
in	<i>t</i>	Time value
in	<i>ts</i>	Time step

Equation (const Element * el)

Constructor using [Element](#) data.

Parameters

in	<i>el</i>	Pointer to Element
-----------	-----------	------------------------------------

Equation (const Element * el, const Vect< T_ > & u, const real_t & time = 0)

Constructor using element data, solution at previous time step and time value.

Parameters

in	<i>el</i>	Pointer to element
in	<i>u</i>	Vect instance containing solution at previous time step
in	<i>time</i>	Time value (Default value is 0)

Equation (const Side * sd, const Vect< T_ > & u, const real_t & time = 0)

Constructor using side data, solution at previous time step and time value.

Parameters

in	<i>sd</i>	Pointer to side
in	<i>u</i>	Vect instance containing solution at previous time step
in	<i>time</i>	Time value (Default value is 0)

7.32.3 Member Function Documentation

void updateBC (const Element & el, const Vect< T_ > & bc)

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< T_ > & bc)

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0)

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< T_ > & b)

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
----	---	--

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEE_ > & be)

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be)

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementNodeVector (const Vect< T_ > & b, LocalVect< T_, NEN_ > & be, int dof)

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only the dega dof is transferred to the local vector

void ElementSideVector (const Vect< T_ > & b, LocalVect< T_, NSE_ > & be)

Localize [Element](#) Vector from a [Vect](#) instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< T_ > & b, int dof_type = NODE_FIELD, int flag = 0)

Localize [Element](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system <p>The resulting local vector can be accessed by attribute ePrev.</p>

Remarks

This member function is to be used if a constructor with [Element](#) was invoked. It uses the [Element](#) pointer `_theElement`

void SideVector (const Vect< T_ > & b)

Localize [Side](#) Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides <p>The resulting local vector can be accessed by attribute ePrev.</p>
----	----------	---

Remarks

This member function is to be used if a constructor with [Side](#) was invoked. It uses the [Side](#) pointer `_theSide`

void ElementNodeCoordinates ()

Localize coordinates of element nodes.
Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the [Side](#) pointer `_theSide`

void SideNodeCoordinates ()

Localize coordinates of side nodes.
Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the [Element](#) pointer `_theElement`

void ElementAssembly ([Matrix< T_ > * A](#))

Assemble element matrix into global one.
Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
-------------------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly ([SkSMatrix< T_ > & A](#))

Assemble element matrix into global one.
Parameters

A	Global matrix stored as an SkSMatrix instance
-------------------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly ([SkMatrix< T_ > & A](#))

Assemble element matrix into global one.
Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly ([SpMatrix](#)< T_ > & A)

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly ([TrMatrix](#)< T_ > & A)

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly ([Matrix](#)< T_ > * A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly ([SkSMatrix](#)< T_ > & A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly ([SkMatrix](#)< T_ > & A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< T_ > & A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< T_ > & A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< T_ > * A)

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix , SkMatrix , SpMatrix)
--	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< T_ > & A)

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< T_ > & A)

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< T_ > & A)

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void ElementAssembly (Vect< T_ > & v)

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The element pointer is given by the global variable theElement

void SideAssembly (Vect< T_ > & v)

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	--

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< T_ > & x, Vect< T_ > & b)

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< T_ > & x, Vect< T_ > & b)

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop)

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to [Mesh](#) instance.

Returns

Reference to [Mesh](#) instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
----	-----------	---

in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
----	----	--

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.33 Estimator Class Reference

To calculate an a posteriori estimator of the solution.

Public Member Functions

- [Estimator](#) ()
Default Constructor.
- [Estimator](#) ([Mesh](#) &m)
Constructor using finite element mesh.
- [~Estimator](#) ()
Destructor.
- void [setError](#) (const [Vect](#)< double > &u)

Calculate error using `Vect` solution vector `u`.

- `double getAverage () const`

Return averaged error.

- `Mesh & getMesh () const`

Return a reference to the finite element mesh.

Public Attributes

- `Vect< double > Err`

Elementwise vector error.

7.33.1 Detailed Description

To calculate an a posteriori estimator of the solution.

This class enables calculating an estimator of a solution in order to evaluate reliability. Estimation uses the so-called Zienkiewicz-Zhu estimator.

7.34 FastMarching2D Class Reference

To run a Fast Marching Method on 2-D structured uniform grids.

Public Member Functions

- `FastMarching2D ()`

Default constructor.

- `FastMarching2D (const Grid &g, Vect< real_t > &ls)`

Constructor using grid and level set function.

- `FastMarching2D (const Grid &g, Vect< real_t > &ls, Vect< real_t > &F)`

Constructor using grid, level set function and velocity to extend.

- `~FastMarching2D ()`

Destructor.

- `void execute ()`

Execute Fast Marching Procedure.

- `void Check ()`

Check distance function.

7.34.1 Detailed Description

To run a Fast Marching Method on 2-D structured uniform grids.

This class enables running a Fast Marching procedure to calculate the signed distance function and extend a given front speed.

7.34.2 Constructor & Destructor Documentation

`FastMarching2D (const Grid & g, Vect< real_t > & ls)`

Constructor using grid and level set function.

Parameters

<code>in</code>	<code>g</code>	Instance of class Grid
<code>in</code>	<code>ls</code>	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.

FastMarching2D (`const Grid & g, Vect< real.t > & ls, Vect< real.t > & F`)

Constructor using grid, level set function and velocity to extend.

Parameters

<code>in</code>	<code>g</code>	Instance of class Grid
<code>in</code>	<code>ls</code>	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided their sign is right.
<code>in</code>	<code>F</code>	Vector containing the front speed at grid nodes. Only values on nodes surrounding the interface are relevant.

7.34.3 Member Function Documentation

void execute ()

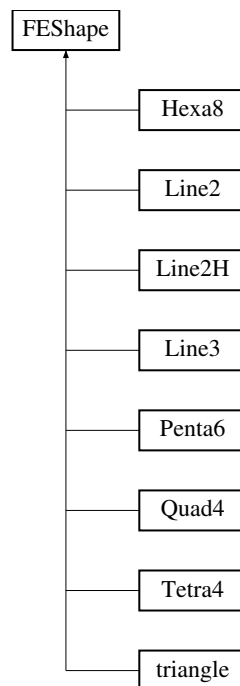
Execute Fast Marching Procedure.

Once this function was called, the vector `ls` used in the constructor will contain the signed distance function and `F` will contain the extended speed.

7.35 FEShape Class Reference

Parent class from which inherit all finite element shape classes.

Inheritance diagram for FEShape:



Public Member Functions

- [FEShape](#) ()
Default Constructor.
- [FEShape](#) (const [Element](#) *el)
Constructor for an element.
- [FEShape](#) (const [Side](#) *sd)
Constructor for a side.
- virtual [~FEShape](#) ()
Destructor.
- [real_t](#) [Sh](#) (size_t i) const
Return shape function of node i at given point.
- [real_t](#) [Sh](#) (size_t i, [Point](#)< [real_t](#) > s) const
Calculate shape function of node i at a given point s .
- [Point](#)< [real_t](#) > [DSh](#) (size_t i) const
Return derivatives of shape function of node i at a given point.
- [real_t](#) [getDet](#) () const
Return determinant of jacobian.
- [Point](#)< [real_t](#) > [getCenter](#) () const
Return coordinates of center of element.
- [Point](#)< [real_t](#) > [getLocalPoint](#) () const
Localize a point in the element.
- [Point](#)< [real_t](#) > [getLocalPoint](#) (const [Point](#)< [real_t](#) > &s) const
Localize a point in the element.

7.35.1 Detailed Description

Parent class from which inherit all finite element shape classes.

7.35.2 Constructor & Destructor Documentation

FEShape (const Element * el)

Constructor for an element.

Parameters

<i>in</i>	<i>el</i>	Pointer to element
-----------	-----------	--------------------

FEShape (const Side * sd)

Constructor for a side.

Parameters

<i>in</i>	<i>sd</i>	Pointer to side
-----------	-----------	-----------------

7.35.3 Member Function Documentation

real_t Sh (size_t i, Point< real_t > s) const

Calculate shape function of node *i* at a given point *s*.

Parameters

<i>in</i>	<i>i</i>	Local node label
<i>in</i>	<i>s</i>	Point in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const

Return derivatives of shape function of node *i* at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

Parameters

<i>in</i>	<i>i</i>	Partial derivative index (1, 2 or 3)
-----------	----------	--------------------------------------

real_t getDet () const

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const

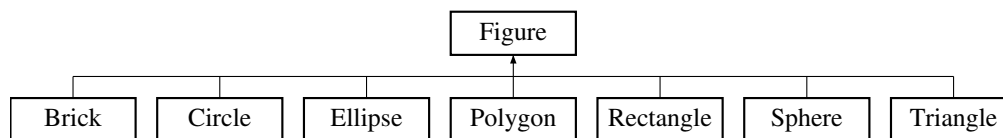
Localize a point in the element.

Return actual coordinates where *s* are coordinates in the reference element.

7.36 Figure Class Reference

To store and treat a figure (or shape) information.

Inheritance diagram for Figure:



Public Member Functions

- [Figure](#) ()
Default constructor.
- [Figure](#) (const [Figure](#) &f)
Copy constructor.
- virtual [~Figure](#) ()
Destructor.
- void [setCode](#) (int code)
Choose a code for the domain defined by the figure.
- virtual double [getSignedDistance](#) (const [Point](#)< double > &p) const
Return signed distance from a given point to current figure.
- [Figure](#) & [operator=](#) (const [Figure](#) &f)
Operator =.
- void [getSignedDistance](#) (const [Grid](#) &g, [Vect](#)< double > &d) const
Calculate signed distance to current figure with respect to grid points.
- double [dLine](#) (const [Point](#)< double > &p, const [Point](#)< double > &a, const [Point](#)< double > &b) const
Compute signed distance from a line.

7.36.1 Detailed Description

To store and treat a figure (or shape) information.

This class is essentially useful to construct data for mesh generators and for distance calculations.

7.36.2 Member Function Documentation

virtual double [getSignedDistance](#) (const [Point](#)< double > & p) const [virtual]

Return signed distance from a given point to current figure.

Parameters

in	<i>p</i>	Point instance from which distance is computed
----	----------	--

Reimplemented in [Polygon](#), [Triangle](#), [Ellipse](#), [Sphere](#), [Circle](#), [Brick](#), and [Rectangle](#).

void [getSignedDistance](#) (const [Grid](#) & g, [Vect](#)< double > & d) const

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector d doesn't need to be sized before invoking this function

```
double dLine ( const Point< double > & p, const Point< double > & a, const Point<  
double > & b ) const
```

Compute signed distance from a line.

Parameters

in	p	Point for which distance is computed
in	a	First vertex of line
in	b	Second vertex of line

Returns

Signed distance

7.37 FMM2D Class Reference

class for the fast marching 2-D algorithm

Inherits FMM.

Public Member Functions

- **FMM2D** (const [Grid](#) &g, [Vect](#)< [real_t](#) > *phi, bool HA)
Constructor.
- void **InitHeap** (Heap &NarrowPt)
- void **solve** ()
Execute Fast Marching Procedure.
- void **Evaluate** (IPoint &pt, int sign)
compute the distance from node to interface
- void **ExtendSpeed** ([Vect](#)< [real_t](#) > &F)
Extend the speed function to the whole grid.
- [real_t](#) **check_error** ()
Check error by comparing with the gradient norm.

7.37.1 Detailed Description

class for the fast marching 2-D algorithm

This class manages the 2-D Fast Marching method

7.37.2 Constructor & Destructor Documentation

FMM2D (const [Grid](#) & g, [Vect](#)< [real_t](#) > * phi, bool HA)

Constructor.

Parameters

in	g	Instance of class Grid
in	phi	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	HA	true if the program must be executed with high accuracy, false otherwise

7.37.3 Member Function Documentation

void InitHeap (Heap & NarrowPt)

Initialize the heap

Parameters

in,out	<i>NarrowPt</i>	Heap containing Narrow points
--------	-----------------	-------------------------------

void Evaluate (IPoint & pt, int sign)

compute the distance from node to interface

Parameters

in	<i>pt</i>	node to treat
in	<i>sign</i>	Node sign

Returns

distance from node pt to interface

void ExtendSpeed (Vect< real.t > & F)

Extend the speed function to the whole grid.

Parameters

in,out	<i>F</i>	Vector containing the speed at interface nodes on input and extended speed at whole grid nodes
--------	----------	--

real.t check_error ()

Check error by comparing with the gradient norm.

This function prints discrete L^2 and Max. errors

7.38 FMM3D Class Reference

class for the 3-D fast marching algorithm

Inherits FMM.

Public Member Functions

- [FMM3D](#) (const [Grid](#) &g, [Vect](#)< [real.t](#) > *phi, bool HA)

Constructor.

- void [InitHeap](#) (Heap &NarrowPt)

Initialize heap.

- void [solve](#) ()

Execute Fast Marching Procedure.

- void [Evaluate](#) (IPoint &pt, int sign)

Compute the distance from node to interface.

- void [ExtendSpeed](#) ([Vect](#)< [real.t](#) > &F)

Extend the speed function to the whole grid.

- [real.t check_error](#) ()

Check error by comparing with the gradient norm.

7.38.1 Detailed Description

class for the 3-D fast marching algorithm

This class manages the 3-D Fast Marching Method

7.38.2 Constructor & Destructor Documentation

FMM3D (const Grid & g, Vect< real_t > * phi, bool HA)

Constructor.

Parameters

in	<i>g</i>	Instance of class Grid
in	<i>phi</i>	Vector containing the level set function at grid nodes. The values are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface but can take any value on other nodes, provided they have the right sign.
in	<i>HA</i>	true if the program must be executed with high accuracy, false otherwise

7.38.3 Member Function Documentation

void InitHeap (Heap & NarrowPt)

Initialize heap.

Parameters

<i>NarrowPt</i>	
-----------------	--

void Evaluate (IPoint & pt, int sign)

Compute the distance from node to interface.

Parameters

in	<i>pt</i>	node to treat
in	<i>sign</i>	the node's sign

Returns

distance from node *pt* to interface

void ExtendSpeed (Vect< real.t > & F)

Extend the speed function to the whole grid.

Parameters

in,out	<i>F</i>	Vector containing the speed at interface nodes on input and extended speed at whole grid nodes
--------	----------	--

real.t check_error ()

Check error by comparing with the gradient norm.

This function prints discrete L^2 and Max. errors

7.39 FMMSolver Class Reference

The Fast Marching Method solver.

Public Member Functions

- [FMMSolver](#) (const [Grid](#) &g, [Vect](#)< [real.t](#) > &phi, bool ha=false)
Constructor.

- `~FMMSolver ()`
Destructor.
- `void solve ()`
Execute the fast marching program.
- `void ExtendSpeed (Vect< real_t > &F)`
Extend speed by Sethian's method.
- `real_t check_error ()`
Return the consistency error of the method.

7.39.1 Detailed Description

The Fast Marching Method solver.

This class enables computing the signed distance function with respect to an interface. It works in 2-D and 3-D on a structured grid. The class is an interface for client. It points to FMM

7.39.2 Constructor & Destructor Documentation

FMMSolver (const Grid & g, Vect< real_t > & phi, bool ha = false)

Constructor.
Parameters

in	<i>g</i>	Instance of class Grid defining the grid on which the distance is computed.
in	<i>phi</i>	Vector containing the level set function at grid nodes. The vector entries are 0 on the interface (from which the distance is computed), positive on one side and negative on the other side. They must contain the signed distance on the nodes surrounding the interface. These values identify by linear interpolation the interface position. The vector entries can take any value on other grid nodes, provided they have the right sign.
in	<i>ha</i>	true if high accuracy FMM is active. The high accuracy version is more accurate but requires more accurate values on the nodes neighbouring the interface.

7.39.3 Member Function Documentation

void ExtendSpeed (Vect< real_t > & F)

Extend speed by Sethian's method.

The method consists in calculating a speed F such that its gradient is orthogonal to the gradient of the level set function

Parameters

in,out	<i>F</i>	Speed function where on input the value of the function is meaningful on the interface. On output F contains the extended speed
--------	----------	---

real_t check_error ()

Return the consistency error of the method.

Consistency is measured by computing the discrete value of the norm of the gradient of the signed distance and subtracting the obtained norm from 1. The absolute value of the result is returned.

7.40 Funct Class Reference

A simple class to parse real valued functions.

Public Member Functions

- `Funct ()`
Default constructor.
- `Funct (string v)`
Constructor for a function of one variable.
- `Funct (string v1, string v2)`
Constructor for a function of two variables.
- `Funct (string v1, string v2, string v3)`
Constructor for a function of three variables.
- `Funct (string v1, string v2, string v3, string v4)`
Constructor for a function of four variables.
- `~Funct ()`
Destructor.
- `double operator() (double x) const`
Operator () to evaluate the function with one variable x
- `double operator() (double x, double y) const`
Operator () to evaluate the function with two variables x, y
- `double operator() (double x, double y, double z) const`
Operator () to evaluate the function with three variables x, y, z
- `double operator() (double x, double y, double z, double t) const`
Operator () to evaluate the function with four variables x, y, z
- `void operator= (string e)`
Operator =.

7.40.1 Detailed Description

A simple class to parse real valued functions.

Functions must have 1, 2, 3 or at most 4 variables.

Warning

Data in the file must be listed in the following order:

```
for x=x_0,...,x_I
  for y=y_0,...,y_J
    for z=z_0,...,z_K
      read v(x,y,z)
```

7.40.2 Constructor & Destructor Documentation

`Funct (string v)`

Constructor for a function of one variable.

Parameters

<code>in</code>	<code>v</code>	Name of the variable
-----------------	----------------	----------------------

Func (string v1, string v2)

Constructor for a function of two variables.

Parameters

<code>in</code>	<code>v1</code>	Name of the first variable
<code>in</code>	<code>v2</code>	Name of the second variable

Func (string v1, string v2, string v3)

Constructor for a function of three variables.

Parameters

<code>in</code>	<code>v1</code>	Name of the first variable
<code>in</code>	<code>v2</code>	Name of the second variable
<code>in</code>	<code>v3</code>	Name of the third variable

Func (string v1, string v2, string v3, string v4)

Constructor for a function of four variables.

Parameters

<code>in</code>	<code>v1</code>	Name of the first variable
<code>in</code>	<code>v2</code>	Name of the second variable
<code>in</code>	<code>v3</code>	Name of the third variable
<code>in</code>	<code>v4</code>	Name of the fourth variable

7.40.3 Member Function Documentation

void operator= (string e)

Operator =.

Define the function by an algebraic expression following regexp rules

Parameters

<code>in</code>	<code>e</code>	Algebraic expression defining the function.
-----------------	----------------	---

7.41 Gauss Class Reference

Calculate data for Gauss integration.

Public Member Functions

- [Gauss](#) ()
Default constructor.
- [Gauss](#) (size_t np)
Constructor using number of [Gauss](#) points.
- void [setTriangle](#) (LocalVect< [real_t](#), 7 > &w, LocalVect< [Point](#)< [real_t](#) >, 7 > &x)
Choose integration on triangle (7-point formula)

- `real_t x (size_t i) const`
Return coordinate of i -th Gauss-Legendre point.
- `const Point< real_t > & xt (size_t i) const`
Return coordinates of points in the reference triangle.
- `real_t w (size_t i) const`
Return weight of i -th Gauss-Legendre point.

7.41.1 Detailed Description

Calculate data for Gauss integration.

7.41.2 Member Function Documentation

`void setTriangle (LocalVect< real_t, 7 > & w, LocalVect< Point< real_t >, 7 > & x)`

Choose integration on triangle (7-point formula)

If this is not selected, `Gauss` integration formula on $[-1,1]$ is calculated.

7.42 Grid Class Reference

To manipulate structured grids.

Public Member Functions

- `Grid ()`
Construct a default grid with 10 intervals in each direction.
- `Grid (real_t xm, real_t xM, size_t npx)`
Construct a 1-D structured grid given its extremal coordinates and number of intervals.
- `Grid (real_t xm, real_t xM, real_t ym, real_t yM, size_t npx, size_t npy)`
Construct a 2-D structured grid given its extremal coordinates and number of intervals.
- `Grid (Point< real_t > m, Point< real_t > M, size_t npx, size_t npy)`
Construct a 2-D structured grid given its extremal coordinates and number of intervals.
- `Grid (real_t xm, real_t xM, real_t ym, real_t yM, real_t zm, real_t zM, size_t npx, size_t npy, size_t npz)`
Construct a 3-D structured grid given its extremal coordinates and number of intervals.
- `Grid (Point< real_t > m, Point< real_t > M, size_t npx, size_t npy, size_t npz)`
Construct a 3-D structured grid given its extremal coordinates and number of intervals.
- `void setXMin (const Point< real_t > &x)`
Set min. coordinates of the domain.
- `void setXMax (const Point< real_t > &x)`
- `void setDomain (real_t xmin, real_t xmax)`
Set Dimensions of the domain: 1-D case.
- `void setDomain (real_t xmin, real_t xmax, real_t ymin, real_t ymax)`
Set Dimensions of the domain: 2-D case.
- `void setDomain (real_t xmin, real_t xmax, real_t ymin, real_t ymax, real_t zmin, real_t zmax)`
Set Dimensions of the domain: 3-D case.
- `void setDomain (Point< real_t > xmin, Point< real_t > xmax)`
Set Dimensions of the domain: 3-D case.
- `const Point< real_t > & getXMin () const`

- Return min. Coordinates of the domain.*
- `const Point< real_t > & getXMax () const`
- Return max. Coordinates of the domain.*
- `void setN (size_t nx, size_t ny=0, size_t nz=0)`
- Set number of grid intervals in the x, y and z-directions.*
- `size_t getNx () const`
- Return number of grid intervals in the x-direction.*
- `size_t getNy () const`
- Return number of grid intervals in the y-direction.*
- `size_t getNz () const`
- Return number of grid intervals in the z-direction.*
- `real_t getHx () const`
- Return grid size in the x-direction.*
- `real_t getHy () const`
- Return grid size in the y-direction.*
- `real_t getHz () const`
- Return grid size in the z-direction.*
- `Point< real_t > getCoord (size_t i) const`
- Return coordinates a point with label i in a 1-D grid.*
- `Point< real_t > getCoord (size_t i, size_t j) const`
- Return coordinates a point with label (i, j) in a 2-D grid.*
- `Point< real_t > getCoord (size_t i, size_t j, size_t k) const`
- Return coordinates a point with label (i, j, k) in a 3-D grid.*
- `real_t getX (size_t i) const`
- Return x-coordinate of point with index i*
- `real_t getY (size_t j) const`
- Return y-coordinate of point with index j*
- `real_t getZ (size_t k) const`
- Return z-coordinate of point with index k*
- `Point2D< real_t > getXY (size_t i, size_t j) const`
- Return coordinates of point with indices (i, j)*
- `Point< real_t > getXYZ (size_t i, size_t j, size_t k) const`
- Return coordinates of point with indices (i, j, k)*
- `real_t getCenter (size_t i) const`
- Return coordinates of center of a 1-D cell with indices i, i+1*
- `Point< real_t > getCenter (size_t i, size_t j) const`
- Return coordinates of center of a 2-D cell with indices (i, j), (i+1, j), (i+1, j+1), (i, j+1)*
- `Point< real_t > getCenter (size_t i, size_t j, size_t k) const`
- Return coordinates of center of a 3-D cell with indices (i, j, k), (i+1, j, k), (i+1, j+1, k), (i, j+1, k), (i, j, k+1), (i+1, j, k+1), (i+1, j+1, k+1), (i, j+1, k+1)*
- `void setCode (string exp, int code)`
- Set a code for some grid points.*
- `void setCode (int side, int code)`
- Set a code for grid points on sides.*
- `int getCode (int side) const`
- Return code for a side number.*
- `int getCode (size_t i, size_t j) const`

- *Return code for a grid point.*
• int `getCode` (size_t i, size_t j, size_t k) const
Return code for a grid point.
- size_t `getDim` () const
Return space dimension.
- void `Deactivate` (size_t i)
Change state of a cell from active to inactive (1-D grid)
- void `Deactivate` (size_t i, size_t j)
Change state of a cell from active to inactive (2-D grid)
- void `Deactivate` (size_t i, size_t j, size_t k)
Change state of a cell from active to inactive (2-D grid)
- int `isActive` (size_t i) const
Say if cell is active or not (1-D grid)
- int `isActive` (size_t i, size_t j) const
Say if cell is active or not (2-D grid)
- int `isActive` (size_t i, size_t j, size_t k) const
Say if cell is active or not (3-D grid)

7.42.1 Detailed Description

To manipulate structured grids.

7.42.2 Constructor & Destructor Documentation

Grid (real_t xm, real_t xM, size_t npx)

Construct a 1-D structured grid given its extremal coordinates and number of intervals.

Parameters

in	<i>xm</i>	Minimal value for x
in	<i>xM</i>	Maximal value for x
in	<i>npx</i>	Number of grid intervals in the x-direction

Grid (real_t xm, real_t xM, real_t ym, real_t yM, size_t npx, size_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Parameters

in	<i>xm</i>	Minimal value for x
in	<i>xM</i>	Maximal value for x
in	<i>ym</i>	Minimal value for y
in	<i>yM</i>	Maximal value for y
in	<i>npx</i>	Number of grid intervals in the x-direction
in	<i>npv</i>	Number of grid intervals in the y-direction

Grid (Point< real_t > m, Point< real_t > M, size_t npx, size_t npy)

Construct a 2-D structured grid given its extremal coordinates and number of intervals.

Parameters

in	<i>m</i>	Minimal coordinate value
in	<i>M</i>	Maximal coordinate value
in	<i>npx</i>	Number of grid intervals in the x-direction
in	<i>npz</i>	Number of grid intervals in the y-direction

Grid (real_t xm, real_t xM, real_t ym, real_t yM, real_t zm, real_t zM, size_t npx, size_t npy, size_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

Parameters

in	<i>xm</i>	Minimal value for x
in	<i>xM</i>	Maximal value for x
in	<i>ym</i>	Minimal value for y
in	<i>yM</i>	Maximal value for y
in	<i>zm</i>	Minimal value for z
in	<i>zM</i>	Maximal value for z
in	<i>npx</i>	Number of grid intervals in the x-direction
in	<i>npy</i>	Number of grid intervals in the y-direction
in	<i>npz</i>	Number of grid intervals in the z-direction

Grid (Point< real_t > m, Point< real_t > M, size_t npx, size_t npy, size_t npz)

Construct a 3-D structured grid given its extremal coordinates and number of intervals.

Parameters

in	<i>m</i>	Minimal coordinate value
in	<i>M</i>	Maximal coordinate value
in	<i>npx</i>	Number of grid intervals in the x-direction
in	<i>npy</i>	Number of grid intervals in the y-direction
in	<i>npz</i>	Number of grid intervals in the z-direction

7.42.3 Member Function Documentation

void setXMin (const Point< real_t > & x)

Set min. coordinates of the domain.

Parameters

in	<i>x</i>	Minimal values of coordinates
----	----------	-------------------------------

void setXMax (const Point< real_t > & x)

Set max. coordinates of the domain.

Parameters

in	<i>x</i>	Maximal values of coordinates
----	----------	-------------------------------

void setDomain (real_t xmin, real_t xmax)

Set Dimensions of the domain: 1-D case.

Parameters

in	<i>xmin</i>	Minimal value of x-coordinate
in	<i>xmax</i>	Maximal value of x-coordinate

void setDomain (real_t xmin, real_t xmax, real_t ymin, real_t ymax)

Set Dimensions of the domain: 2-D case.

Parameters

in	<i>xmin</i>	Minimal value of x-coordinate
in	<i>xmax</i>	Maximal value of x-coordinate
in	<i>ymin</i>	Minimal value of y-coordinate
in	<i>ymax</i>	Maximal value of y-coordinate

void setDomain (real_t xmin, real_t xmax, real_t ymin, real_t ymax, real_t zmin, real_t zmax)

Set Dimensions of the domain: 3-D case.

Parameters

in	<i>xmin</i>	Minimal value of x-coordinate
in	<i>xmax</i>	Maximal value of x-coordinate
in	<i>ymin</i>	Minimal value of y-coordinate
in	<i>ymax</i>	Maximal value of y-coordinate
in	<i>zmin</i>	Minimal value of z-coordinate
in	<i>zmax</i>	Maximal value of z-coordinate

void setDomain (Point< real_t > xmin, Point< real_t > xmax)

Set Dimensions of the domain: 3-D case.

Parameters

in	<i>xmin</i>	Minimal coordinate value
in	<i>xmax</i>	Maximal coordinate value

void setN (size_t nx, size_t ny = 0, size_t nz = 0)

Set number of grid intervals in the x, y and z-directions.

Number of points is the number of intervals plus one in each direction

Parameters

in	<i>nx</i>	Number of grid intervals in the x-direction
in	<i>ny</i>	Number of grid intervals in the y-direction (Default=0: 1-D grid)
in	<i>nz</i>	Number of grid intervals in the z-direction (Default=0: 1-D or 2-D grid)

Remarks

: The size of the grid (*xmin* and *xmax*) must have been defined before.

size_t getNy () const

Return number of grid intervals in the y-direction.

ny=0 for 1-D domains (segments)

size_t getNz () const

Return number of grid intervals in the z-direction.

`nz=0` for 1-D (segments) and 2-D domains (rectangles)

void setCode (string exp, int code)

Set a code for some grid points.

Parameters

<code>in</code>	<code>exp</code>	Regular expression that determines the set of grid points on which the code is applied.
<code>in</code>	<code>code</code>	Code to assign.

void setCode (int side, int code)

Set a code for grid points on sides.

Parameters

<code>in</code>	<code>side</code>	Side for which code is assigned. Possible values are: <code>MIN_X</code> , <code>MAX_X</code> , <code>MIN_Y</code> , <code>MAX_Y</code> , <code>MIN_Z</code> , <code>MAX_Z</code>
<code>in</code>	<code>code</code>	Code to assign.

int getCode (int side) const

Return code for a side number.

Parameters

<code>in</code>	<code>side</code>	Side for which code is returned. Possible values are: <code>MIN_X</code> , <code>MAX_X</code> , <code>MIN_Y</code> , <code>MAX_Y</code> , <code>MIN_Z</code> , <code>MAX_Z</code>
-----------------	-------------------	---

int getCode (size_t i, size_t j) const

Return code for a grid point.

Parameters

<code>in</code>	<code>i</code>	<code>i</code> -th index for node for which code is to be returned.
<code>in</code>	<code>j</code>	<code>j</code> -th index for node for which code is to be returned.

int getCode (size_t i, size_t j, size_t k) const

Return code for a grid point.

Parameters

<code>in</code>	<code>i</code>	<code>i</code> -th index for node for which code is to be returned.
<code>in</code>	<code>j</code>	<code>j</code> -th index for node for which code is to be returned.
<code>in</code>	<code>k</code>	<code>k</code> -th index for node for which code is to be returned.

void Deactivate (size_t i)

Change state of a cell from active to inactive (1-D grid)

Parameters

<code>in</code>	<code>i</code>	grid cell to remove
-----------------	----------------	---------------------

void Deactivate (`size_t i`, `size_t j`)

Change state of a cell from active to inactive (2-D grid)

Parameters

<code>in</code>	<code>i</code>	i-th index for grid cell to remove. If this value is 0, all cells (*,j) are deactivated
<code>in</code>	<code>j</code>	j-th index for grid cell to remove. If this value is 0, all cells (i,*) are deactivated

Remarks

if i and j have value 0 all grid cells are deactivated !!

void Deactivate (`size_t i`, `size_t j`, `size_t k`)

Change state of a cell from active to inactive (2-D grid)

Parameters

<code>in</code>	<code>i</code>	i-th index for grid cell to remove. If this value is 0, all cells (*,j,k) are deactivated
<code>in</code>	<code>j</code>	j-th index for grid cell to remove. If this value is 0, all cells (i,*,k) are deactivated
<code>in</code>	<code>k</code>	k-th index for grid cell to remove. If this value is 0, all cells (i,j,*) are deactivated

int isActive (`size_t i`) const

Say if cell is active or not (1-D grid)

Parameters

<code>in</code>	<code>i</code>	Index of cell
-----------------	----------------	---------------

Returns

1 if cell is active, 0 if not

int isActive (`size_t i`, `size_t j`) const

Say if cell is active or not (2-D grid)

Parameters

<code>in</code>	<code>i</code>	i-th index of cell
<code>in</code>	<code>j</code>	j-th index of cell

Returns

1 if cell is active, 0 if not

int isActive (`size_t i`, `size_t j`, `size_t k`) const

Say if cell is active or not (3-D grid)

Parameters

in	i	i-th index of cell
in	j	j-th index of cell
in	k	k-th index of cell

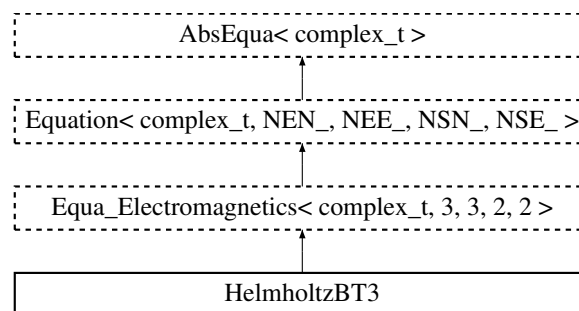
Returns

1 if cell is active, 0 if not

7.43 HelmholtzBT3 Class Reference

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.

Inheritance diagram for HelmholtzBT3:



Public Member Functions

- [HelmholtzBT3](#) ()
Default Constructor.
- [HelmholtzBT3](#) ([Element](#) *el)
Constructor using element data.
- [HelmholtzBT3](#) ([Side](#) *sd)
Constructor using side data.
- [~HelmholtzBT3](#) ()
Destructor.
- void [LHS](#) ([real_t](#) wave_nb)
Add element Left-Hand Side.
- void [BoundaryRHS](#) ([UserData](#)< [complex_t](#) > &ud)
Add element Right-Hand Side using a UserData instance.
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< [complex_t](#) > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< [complex_t](#) > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof.type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< [complex_t](#) > &b)
Localize Element Vector from a Vect instance.
- void [ElementNodeVector](#) (const [Vect](#)< [complex_t](#) > &b, [LocalVect](#)< [complex_t](#), [NEE_](#) > &be)
Localize Element Vector from a Vect instance.

- void `ElementNodeVector` (const `Vect< complex_t >` &b, `LocalVect< complex_t, NEN_ >` &be, int dof)
Localize Element Vector from a Vect instance.
- void `ElementNodeVectorSingleDOF` (const `Vect< complex_t >` &b, `LocalVect< complex_t, NEN_ >` &be)
Localize Element Vector from a Vect instance.
- void `ElementSideVector` (const `Vect< complex_t >` &b, `LocalVect< complex_t, NSE_ >` &be)
Localize Element Vector from a Vect instance.
- void `ElementVector` (const `Vect< complex_t >` &b, int dof_type=NODE.FIELD, int flag=0)
Localize Element Vector.
- void `SideVector` (const `Vect< complex_t >` &b)
Localize Side Vector.
- void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix< complex_t >` *A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix< complex_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< complex_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< complex_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< complex_t >` &A)
Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< complex_t >` &v)
Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< complex_t >` *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix< complex_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< complex_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< complex_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix< complex_t >` &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix< complex_t >` *A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix< complex_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< complex_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix< complex_t >` &A)
Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect< complex_t >` &v)

- Assemble side (edge or face) vector into global one.*

 - void `AxbAssembly` (const `Element` &el, const `Vect`< `complex_t` > &x, `Vect`< `complex_t` > &b)
- Assemble product of element matrix by element vector into global vector.*

 - void `AxbAssembly` (const `Side` &sd, const `Vect`< `complex_t` > &x, `Vect`< `complex_t` > &b)
- Assemble product of side matrix by side vector into global vector.*

 - size_t `getNbNodes` () const
- Return number of element nodes.*

 - size_t `getNbEq` () const
- Return number of element equations.*

 - `complex_t` * `A` ()
- Return element matrix as a C-array.*

 - `complex_t` * `sA` ()
- Return side matrix as a C-array.*

 - `complex_t` * `b` ()
- Return element right-hand side as a C-array.*

 - `complex_t` * `sb` ()
- Return side right-hand side as a C-array.*

 - `complex_t` * `Prev` ()
- Return element matrix as a C-array.*

 - `LocalMatrix`< `complex_t`, `NEE_`, `NEE_` > & `EA` ()
- Return element matrix as a LocalMatrix instance.*

 - `LocalMatrix`< `complex_t`, `NSE_`, `NSE_` > & `SA` ()
- Return side matrix as a LocalMatrix instance.*

 - `LocalVect`< `complex_t`, `NEE_` > & `Eb` ()
- Return element right-hand side as a LocalVect instance.*

 - `LocalVect`< `complex_t`, `NEE_` > & `Ep` ()
- Return element matrix as a C-array.*

 - void `setInitialSolution` (const `Vect`< `complex_t` > &u)
- Set initial solution (previous time step)*

 - `real_t` `setMaterialProperty` (const string &exp, const string &prop)
- Define a material property by an algebraic expression.*

 - void `setMesh` (class `Mesh` &m)
- Define mesh and renumber DOFs after removing imposed ones.*

 - `Mesh` & `getMesh` () const
- Return reference to Mesh instance.*

 - `LinearSolver`< `complex_t` > & `getLinearSolver` ()
- Return reference to linear solver instance.*

 - void `setSolver` (int ls, int pc=`IDENT_PREC`)
- Choose solver for the linear system.*

 - int `solveEigenProblem` (int nb_eigv, bool g=false)
- Compute eigenvalues and eigenvectors.*

 - `real_t` `getEigenValue` (int n) const
- Return the n-th eigenvalue.*

 - void `getEigenvector` (int n, `Vect`< `real_t` > &v) const
- Store the eigenvector corresponding to a given eigenvalue.*

 - class `Eigen` & `getEigenSolver` ()
- Return reference to eigenproblem solver.*

Protected Member Functions

- void [MagneticPermeability](#) (const [real_t](#) &mu)
Set (constant) magnetic permeability.
- void [MagneticPermeability](#) (const string &exp)
Set magnetic permeability given by an algebraic expression.
- void [ElectricConductivity](#) (const [real_t](#) &sigma)
Set (constant) electric conductivity.
- void [ElectricConductivity](#) (const string &exp)
set electric conductivity given by an algebraic expression
- void [ElectricResistivity](#) (const [real_t](#) &rho)
Set (constant) electric resistivity.
- void [ElectricResistivity](#) (const string &exp)
Set electric resistivity given by an algebraic expression.
- void [setMaterial](#) ()
Set material properties.
- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.43.1 Detailed Description

Builds finite element arrays for Helmholtz equations in a bounded media using 3-Node triangles.
Problem being formulated in time harmonics, the solution is complex valued.

7.43.2 Member Function Documentation

void updateBC (const [Element](#) & el, const [Vect](#)< [complex_t](#) > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	el	Reference to current element instance
in	bc	Vector that contains imposed values at all DOFs

void updateBC (const [Vect](#)< [complex_t](#) > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	bc	Vector that contains imposed values at all DOFs
--------------------	--------------------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = [NODE_DOF](#), int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< complex_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< complex_t > & b, LocalVect< complex_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< complex_t > & b, LocalVect< complex_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< complex_t > & b, LocalVect<  
complex_t , NEN_ > & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< complex_t > & b, LocalVect< complex_t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< complex_t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< complex_t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Element` pointer `_theElement`

void ElementAssembly (Matrix< complex_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMatrix</code> instance
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< complex_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< complex_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< complex_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< complex_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< complex_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< complex.t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< complex.t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< complex.t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< complex.t > & x, Vect< complex.t > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)

out	<i>b</i>	Global vector to add (Vect instance)
-----	----------	--------------------------------------

void AxbAssembly (const Side & sd, const Vect< complex.t > & x, Vect< complex.t > & b) [inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real.t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

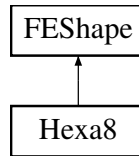
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.44 Hexa8 Class Reference

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

Inheritance diagram for Hexa8:



Public Member Functions

- [Hexa8](#) ()
Default Constructor.
- [Hexa8](#) (const [Element](#) *el)
Constructor when data of [Element](#) el are given.
- [~Hexa8](#) ()
Destructor.
- void [setLocal](#) (const [Point](#)< [real_t](#) > &s)
Initialize local point coordinates in element.
- [Point](#)< [real_t](#) > [DSH](#) (size_t i)
Return x , y and z partial derivatives of shape function of node i at a given point.
- void [atGauss1](#) ([LocalVect](#)< [Point](#)< [real_t](#) >, 8 > &dsh, [real_t](#) &w)
Calculate shape function derivatives and integration weights for 1-point [Gauss](#) rule.
- void [atGauss2](#) ([LocalMatrix](#)< [Point](#)< [real_t](#) >, 8, 8 > &dsh, [LocalVect](#)< [real_t](#), 8 > &w)
Calculate shape function derivatives and integration weights for 2x2x2-point [Gauss](#) rule.
- [real_t](#) [getMaxEdgeLength](#) () const
Return maximal edge length.
- [real_t](#) [getMinEdgeLength](#) () const
Return minimal edge length.
- [real_t](#) [Sh](#) (size_t i) const
Return shape function of node i at given point.
- [real_t](#) [Sh](#) (size_t i, [Point](#)< [real_t](#) > s) const
Calculate shape function of node i at a given point s .
- [Point](#)< [real_t](#) > [DSH](#) (size_t i) const
Return derivatives of shape function of node i at a given point.
- [real_t](#) [getDet](#) () const
Return determinant of jacobian.
- [Point](#)< [real_t](#) > [getCenter](#) () const
Return coordinates of center of element.
- [Point](#)< [real_t](#) > [getLocalPoint](#) () const

Localize a point in the element.

- `Point< real_t > getLocalPoint (const Point< real_t > &s) const`

Localize a point in the element.

7.44.1 Detailed Description

Defines a three-dimensional 8-node hexahedral finite element using Q1-isoparametric interpolation.

The reference element is the cube $[-1,1]*[-1,1]*[-1,1]$. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function **getLocal(s)** must be invoked.

7.44.2 Member Function Documentation

void setLocal (const Point< real_t > & s)

Initialize local point coordinates in element.

Parameters

<code>in</code>	<code>s</code>	<code>Point</code> in the reference element This function computes jacobian, shape functions and their partial derivatives at s. Other member functions only return these values.
-----------------	----------------	---

Point<real_t> DSh (size_t i)

Return x, y and z partial derivatives of shape function of node i at a given point.

Member function *setLocal* must have been called before in order to calculate relevant quantities.

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

<code>in</code>	<code>i</code>	Local node label
<code>in</code>	<code>s</code>	<code>Point</code> in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const [inherited]

Return derivatives of shape function of node i at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function *setLocal()* must have been called before in order to calculate relevant quantities.

Parameters

<code>in</code>	<code>i</code>	Partial derivative index (1, 2 or 3)
-----------------	----------------	--------------------------------------

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

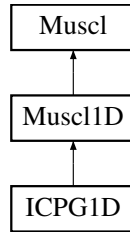
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.45 ICPG1D Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.

Inheritance diagram for ICPG1D:



Public Types

- enum **Method** {
FIRST_ORDER_METHOD = 0,
MULTI_SLOPE_Q_METHOD = 1,
MULTI_SLOPE_M_METHOD = 2,
M_FORCE_WORD = 0xFFFFFFFF }

Enumeration for flux choice.

- enum **Limiter** {
MINMOD_LIMITER = 0,
VANLEER_LIMITER = 1,
SUPERBEE_LIMITER = 2,
VANALBADA_LIMITER = 3,
MAX_LIMITER = 4,
L_FORCE_WORD = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum **SolverType** {
ROE_SOLVER = 0,
VFROE_SOLVER = 1,
LF_SOLVER = 2,
RUSANOV_SOLVER = 3,
HLL_SOLVER = 4,
HLLC_SOLVER = 5,
MAX_SOLVER = 6,
FORCE_WORD = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `ICPG1D (Mesh &ms)`
Constructor using `Mesh` instance.
- `ICPG1D (Mesh &ms, Vect< real_t > &r, Vect< real_t > &v, Vect< real_t > &p)`
Constructor using mesh and initial data.
- `~ICPG1D ()`
Destructor.
- `void setReconstruction ()`
Set reconstruction from class `Muscl`.
- `real_t runOneTimeStep ()`
Advance one time step.
- `void Forward (const Vect< real_t > &flux, Vect< real_t > &field)`
Add flux to field.
- `void setSolver (SolverType solver)`
Choose solver type.
- `void setGamma (real_t gamma)`
Set value of constant `Gamma` for gases.
- `void setCv (real_t Cv)`
Set value of `Cv` (specific heat at constant volume)
- `void setCp (real_t Cp)`
Set value of `Cp` (specific heat at constant pressure)
- `void setKappa (real_t Kappa)`
Set value of constant `Kappa`.
- `real_t getGamma () const`
Return value of constant `Gamma`.
- `real_t getCv () const`
Return value of `Cv` (specific heat at constant volume)
- `real_t getCp () const`
Return value of `Cp` (specific heat at constant pressure)
- `real_t getKappa () const`
Return value of constant `Kappa`.
- `void getMomentum (Vect< real_t > &m) const`
Get vector of momentum at elements.
- `void getInternalEnergy (Vect< real_t > &ie) const`
Get vector of internal energy at elements.
- `void getTotalEnergy (Vect< real_t > &te) const`
Get vector of total energy at elements.
- `void getSoundSpeed (Vect< real_t > &s) const`
Get vector of sound speed at elements.
- `void getMach (Vect< real_t > &m) const`
Get vector of elementwise Mach number.
- `void setInitialCondition_shock_tube (const LocalVect< real_t, 3 > &BcG, const LocalVect< real_t, 3 > &BcD, real_t x0)`
Initial condition corresponding to the shock tube.
- `void setInitialCondition (const LocalVect< real_t, 3 > &u)`
A constant initial condition.

- void `setBC` (const `Side` &sd, `real_t` u)
Assign a boundary condition as a constant to a given side.
- void `setBC` (int code, `real_t` a)
Assign a boundary condition value.
- void `setBC` (`real_t` a)
Assign a boundary condition value.
- void `setBC` (const `Side` &sd, const `LocalVect`< `real_t`, 3 > &u)
Assign a Dirichlet boundary condition vector.
- void `setBC` (int code, const `LocalVect`< `real_t`, 3 > &U)
Assign a Dirichlet boundary condition vector.
- void `setBC` (const `LocalVect`< `real_t`, 3 > &u)
Assign a Dirichlet boundary condition vector.
- void `setInOutflowBC` (const `Side` &sd, const `LocalVect`< `real_t`, 3 > &u)
Impose a constant inflow or outflow boundary condition on a given side.
- void `setInOutflowBC` (int code, const `LocalVect`< `real_t`, 3 > &u)
Impose a constant inflow or outflow boundary condition on sides with a given code.
- void `setInOutflowBC` (const `LocalVect`< `real_t`, 3 > &u)
Impose a constant inflow or outflow boundary condition on boundary sides.
- `real_t` `getMeanLength` () const
Return mean length.
- `real_t` `getMaximumLength` () const
Return maximal length.
- `real_t` `getMinimumLength` () const
Return mimal length.
- `real_t` `getTauLim` () const
Return mean length.
- void `print_mesh_stat` ()
Output mesh information.
- void `setTimeStep` (`real_t` dt)
Assign time step value.
- `real_t` `getTimeStep` () const
Return time step value.
- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getCFL` () const
Return CFL value.
- void `setReferenceLength` (`real_t` dx)
Assign reference length value.
- `real_t` `getReferenceLength` () const
Return reference length.
- `Mesh` & `getMesh` () const
Return reference to `Mesh` instance.
- void `setVerbose` (int v)
Set verbosity parameter.
- bool `setReconstruction` (const `Vect`< `real_t` > &U, `Vect`< `real_t` > &LU, `Vect`< `real_t` > &RU, size_t dof)
Function to reconstruct by the `Muscl` method.

- void `setMethod` (const `Method` &s)
Choose a flux solver.
- void `setSolidZoneCode` (int c)
Choose a code for solid zone.
- bool `getSolidZone` () const
Return flag for presence of solid zones.
- int `getSolidZoneCode` () const
Return code of solid zone, 0 if this one is not present.
- void `setLimiter` (`Limiter` l)
Choose a flux limiter.

7.45.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 1-D.
Solution method is a second-order MUSCL Finite Volume scheme

7.45.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method
MULTI_SLOPE_Q_METHOD Multislope Q method
MULTI_SLOPE_M_METHOD Multislope M method
M_FORCE_WORD for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver
VFROE_SOLVER Finite Volume Roe solver
LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.45.3 Constructor & Destructor Documentation

ICPG1D (Mesh & ms, Vect< real.t > & r, Vect< real.t > & v, Vect< real.t > & p)

Constructor using mesh and initial data.

Parameters

in	<i>ms</i>	Reference to Mesh instance
in	<i>r</i>	Vector containing initial (elementwise) density
in	<i>v</i>	Vector containing initial (elementwise) velocity
in	<i>p</i>	Vector containing initial (elementwise) pressure

7.45.4 Member Function Documentation

void Forward (const Vect< real.t > & flux, Vect< real.t > & field)

Add flux to field.

If this function is used, the user must call `getFlux` himself

Parameters

in	<i>flux</i>	Vector containing fluxes at sides (points)
out	<i>field</i>	Vector containing solution vector

void getMomentum (Vect< real.t > & m) const

Get vector of momentum at elements.

Parameters

in,out	<i>m</i>	Vect instance that contains on output element momentum
--------	----------	--

void getInternalEnergy (Vect< real.t > & ie) const

Get vector of internal energy at elements.

Parameters

in,out	<i>ie</i>	Vect instance that contains on output element internal energy
--------	-----------	---

void getTotalEnergy (Vect< real.t > & te) const

Get vector of total energy at elements.

Parameters

in,out	<i>te</i>	Vect instance that contains on output element total energy
--------	-----------	--

void getSoundSpeed (Vect< real.t > & s) const

Get vector of sound speed at elements.

Parameters

in,out	<i>s</i>	Vect instance that contains on output element sound speed
--------	----------	---

void getMach (Vect< real_t > & m) const

Get vector of elementwise Mach number.

Parameters

in,out	<i>m</i>	Vect instance that contains on output element Mach number
--------	----------	---

void setInitialCondition (const LocalVect< real_t, 3 > & u)

A constant initial condition.

Parameters

in	<i>u</i>	LocalVect instance containing density, velocity and pressure
----	----------	--

void setBC (const Side & sd, real_t u)

Assign a boundary condition as a constant to a given side.

Parameters

in	<i>sd</i>	Side to which the value is assigned
in	<i>u</i>	Value to assign

void setBC (int code, real_t a)

Assign a boundary condition value.

Parameters

in	<i>code</i>	Code value to which boundary condition is assigned
in	<i>a</i>	Value to assign to sides that have code code

void setBC (real_t a)

Assign a boundary condition value.

Parameters

in	<i>a</i>	Value to assign to all boundary sides
----	----------	---------------------------------------

void setBC (const Side & sd, const LocalVect< real_t, 3 > & u)

Assign a Dirichlet boundary condition vector.

Parameters

in	<i>sd</i>	Side instance to which the values are assigned
in	<i>u</i>	LocalVect instance that contains values to assign to the side

void setBC (int code, const LocalVect< real_t, 3 > & U)

Assign a Dirichlet boundary condition vector.

Parameters

in	<i>code</i>	Side code for which the values are assigned
in	<i>U</i>	LocalVect instance that contains values to assign to sides with <i>code</i>

void setBC (const LocalVect< real_t, 3 > & u)

Assign a Dirichlet boundary condition vector.

Parameters

in	<i>u</i>	LocalVect instance that contains values to assign to all boundary sides
----	----------	---

void setInOutflowBC (const Side & sd, const LocalVect< real_t, 3 > & u)

Impose a constant inflow or outflow boundary condition on a given side.

Parameters

in	<i>sd</i>	Instance of Side on which the condition is prescribed
in	<i>u</i>	LocalVect instance that contains values to assign to the side

void setInOutflowBC (int code, const LocalVect< real_t, 3 > & u)

Impose a constant inflow or outflow boundary condition on sides with a given code.

Parameters

in	<i>code</i>	Value of code for which the condition is prescribed
in	<i>u</i>	LocalVect instance that contains values to assign to the sides

void setInOutflowBC (const LocalVect< real_t, 3 > & u)

Impose a constant inflow or outflow boundary condition on boundary sides.

Parameters

in	<i>u</i>	LocalVect instance that contains values to assign to the sides
----	----------	--

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	<i>dx</i>	Value of reference length
----	-----------	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	<i>v</i>	Value of verbosity parameter
----	----------	------------------------------

bool setReconstruction (const Vect< real.t > & U, Vect< real.t > & LU, Vect< real.t > & RU, size.t dof) [inherited]

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	<i>s</i>	Solver to choose
----	----------	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

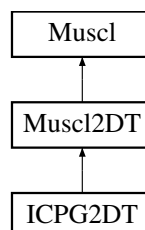
Parameters

in	<i>l</i>	Limiter to choose
----	----------	-------------------

7.46 ICPG2DT Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.

Inheritance diagram for ICPG2DT:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `ICPG2DT` (`Mesh` &ms)
Constructor using mesh instance.
- `ICPG2DT` (`Mesh` &ms, `Vect`< `real_t` > &r, `Vect`< `real_t` > &v, `Vect`< `real_t` > &p)
Constructor using mesh and initial data.
- `~ICPG2DT` ()
Destructor.
- void `setReconstruction` ()
Reconstruct.
- `real_t` `runOneTimeStep` ()
Advance one time step.
- void `Forward` (const `Vect`< `real_t` > &Flux, `Vect`< `real_t` > &Field)
Add Flux to Field.
- `real_t` `getFlux` ()
Get flux.
- void `setSolver` (`SolverType` s)
Choose solver.
- void `setGamma` (`real_t` gamma)
Set Gamma value.
- void `setCv` (`real_t` Cv)
Set value of heat capacity at constant volume.
- void `setCp` (`real_t` Cp)

- Set value of heat capacity at constant pressure.
- void `setKappa` (`real_t` Kappa)
 - Set Kappa value.
- `real_t` `getGamma` () const
 - Return value of Gamma.
- `real_t` `getCv` () const
 - Return value of heat capacity at constant volume.
- `real_t` `getCp` () const
 - Return value of heat capacity at constant pressure.
- `real_t` `getKappa` () const
 - Return value of Kappa.
- `Mesh` & `getMesh` ()
 - Return reference to mesh instance.
- void `getMomentum` (`Vect`< `real_t` > &m) const
 - Calculate elementwise momentum.
- void `getInternalEnergy` (`Vect`< `real_t` > &e) const
 - Calculate elementwise internal energy.
- void `getTotalEnergy` (`Vect`< `real_t` > &e) const
 - Return elementwise total energy.
- void `getSoundSpeed` (`Vect`< `real_t` > &s) const
 - Return elementwise sound speed.
- void `getMach` (`Vect`< `real_t` > &m) const
 - Return elementwise Mach number.
- void `setBC` (const `Side` &sd, `real_t` a)
 - Prescribe a constant boundary condition at given side.
- void `setBC` (int code, `real_t` a)
 - Prescribe a constant boundary condition for a given code.
- void `setBC` (`real_t` u)
 - Prescribe a constant boundary condition on all boundary sides.
- void `setBC` (const `Side` &sd, const `LocalVect`< `real_t`, 4 > &u)
 - Prescribe a constant boundary condition at a given side.
- void `setBC` (int code, const `LocalVect`< `real_t`, 4 > &u)
 - Prescribe a constant boundary condition for a given code.
- void `setBC` (const `LocalVect`< `real_t`, 4 > &u)
 - Prescribe a constant boundary condition at all boundary sides.
- `real_t` `getR` (size_t i) const
 - Return density at given element label.
- `real_t` `getV` (size_t i, size_t j) const
- `real_t` `getP` (size_t i) const
 - Return pressure at given element label.
- bool `setReconstruction` (const `Vect`< `real_t` > &U, `Vect`< `real_t` > &LU, `Vect`< `real_t` > &RU, size_t dof)
 - Function to reconstruct by the *Muscl* method.
- void `setTimeStep` (`real_t` dt)
 - Assign time step value.
- `real_t` `getTimeStep` () const
 - Return time step value.

- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getCFL` () const
Return CFL value.
- void `setReferenceLength` (`real_t` dx)
Assign reference length value.
- `real_t` `getReferenceLength` () const
Return reference length.
- `Mesh` & `getMesh` () const
Return reference to `Mesh` instance.
- void `setVerbose` (int v)
Set verbosity parameter.
- void `setMethod` (const `Method` &s)
Choose a flux solver.
- void `setSolidZoneCode` (int c)
Choose a code for solid zone.
- bool `getSolidZone` () const
Return flag for presence of solid zones.
- int `getSolidZoneCode` () const
Return code of solid zone, 0 if this one is not present.
- void `setLimiter` (`Limiter` l)
Choose a flux limiter.

Protected Member Functions

- void `Initialize` ()
Construction of normals to sides.

7.46.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 2-D.
Solution method is a second-order MUSCL Finite Volume scheme on triangles

7.46.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method

MULTI_SLOPE_Q_METHOD Multislope Q method

MULTI_SLOPE_M_METHOD Multislope M method

M_FORCE_WORD for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatabity reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver
VFROE_SOLVER Finite Volume Roe solver
LF_SOLVER LF solver
RUSANOV_SOLVER Rusanov solver
HLL_SOLVER HLL solver
HLLC_SOLVER HLLC solver
MAX_SOLVER Max solver
FORCE_WORD for compatabity reason, gcc may return a warning anyway

7.46.3 Constructor & Destructor Documentation

ICPG2DT (Mesh & ms, Vect< real_t > & r, Vect< real_t > & v, Vect< real_t > & p)

Constructor using mesh and initial data.

Parameters

in	<i>ms</i>	Mesh instance
in	<i>r</i>	Initial density vector (as instance of Vect)
in	<i>v</i>	Initial velocity vector (as instance of Vect)
in	<i>p</i>	Initial pressure vector (as instance of Vect)

7.46.4 Member Function Documentation

void setReconstruction ()

Reconstruct.

exit(3) if reconstruction fails

void Forward (const Vect< real_t > & Flux, Vect< real_t > & Field)

Add Flux to Field.

If this function is used, the function getFlux must be called

void setSolver (SolverType s)

Choose solver.

Parameters

in	s	Index of solver in the enumerated variable SolverType Available values are: ROE_SOLVER, VFROE_SOLVER, LF_SOLVER, RUSANOV_SOLVER, HLL_SOLVER, HLLC_SOLVER, MAX_SOLVER
-----------	----------	--

void setBC (const Side & sd, real_t a)

Prescribe a constant boundary condition at given side.

Parameters

in	sd	Reference to Side instance
in	a	Value to prescribe

void setBC (int code, real_t a)

Prescribe a constant boundary condition for a given code.

Parameters

in	code	Code for which value is imposed
in	a	Value to prescribe

void setBC (real_t u)

Prescribe a constant boundary condition on all boundary sides.

Parameters

in	u	Value to prescribe
-----------	----------	--------------------

void setBC (const Side & sd, const LocalVect< real_t, 4 > & u)

Prescribe a constant boundary condition at a given side.

Parameters

in	sd	Reference to Side instance
in	u	Vector (instance of class LocalVect) with as components the constant values to prescribe for the four fields (x, vx, vy, p)

void setBC (int code, const LocalVect< real_t, 4 > & u)

Prescribe a constant boundary condition for a given code.

Parameters

in	code	Code for which value is imposed
in	u	Vector (instance of class LocalVect) with as components the constant values to prescribe for the four fields (x, vx, vy, p)

void setBC (const LocalVect< real_t, 4 > & u)

Prescribe a constant boundary condition at all boundary sides.

Parameters

in	<i>u</i>	Vector (instance of class LocalVect) with as components the constant values to prescribe for the four fields (r, vx, vy, p)
----	----------	--

real_t getR (size_t i) const

Return density at given element label.

Parameters

in	<i>i</i>	Element label
----	----------	-------------------------------

real_t getV (size_t i, size_t j) const

Return velocity at given element label

Parameters

in	<i>i</i>	Element label
in	<i>j</i>	component index (1 or 2)

real_t getP (size_t i) const

Return pressure at given element label.

Parameters

in	<i>i</i>	Element label
----	----------	-------------------------------

bool setReconstruction (const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t > & RU, size_t dof) [inherited]Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void Initialize () [protected], [inherited]

Construction of normals to sides.

Convention: for a given side, getPtrElement(1) is the left element and getPtrElement(2) is the right element. The normal goes from left to right. For boundary sides, the normal points outward.

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
-----------	------------------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
-----------	-------------------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	<i>dx</i>	Value of reference length
-----------	------------------	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	<i>v</i>	Value of verbosity parameter
-----------	-----------------	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	<i>s</i>	Solver to choose
-----------	-----------------	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

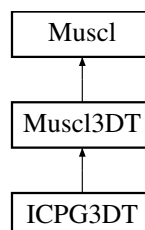
Parameters

in	<i>l</i>	Limiter to choose
-----------	-----------------	-------------------

7.47 ICPG3DT Class Reference

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D.

Inheritance diagram for ICPG3DT:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }
Enumeration for flux choice.
- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }
Enumeration of flux limiting methods.
- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }
Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `ICPG3DT` (`Mesh` &ms)
Constructor using mesh data.
- `ICPG3DT` (`Mesh` &ms, `Vect`< `real_t` > &r, `Vect`< `real_t` > &v, `Vect`< `real_t` > &p)
Constructor using mesh and initial data.
- `~ICPG3DT` ()
Destructor.
- void `setReconstruction` ()
Reconstruct.
- `real_t` `runOneTimeStep` ()
Advance one time step.
- void `Forward` (const `Vect`< `real_t` > &flux, `Vect`< `real_t` > &field)
Add flux to field.
- `real_t` `getFlux` ()
Return flux.
- void `setReferenceLength` (`real_t` dx)
Assign a reference length.
- void `setTimeStep` (`real_t` dt)
Assign a time step.
- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getReferenceLength` () const

- Return reference length.*

 - `real_t getTimeStep ()` const

Return time step.

 - `real_t getCFL ()` const

Return CFL.

 - void `setGamma (real_t gamma)`

Set γ value.

 - void `setCv (real_t Cv)`

Set value of C_V (Heat capacity at constant volume)

 - void `setCp (real_t Cp)`

Set value of C_p (Heat capacity at constant pressure)

 - void `setKappa (real_t Kappa)`

Set Kappa value.

 - `real_t getGamma ()` const

Return value of γ .

 - `real_t getCv ()` const

Return value of C_V (Heat capacity at constant volume)

 - `real_t getCp ()` const

Return value of C_p (Heat capacity at constant pressure)

 - `real_t getKappa ()` const

Return value of κ .

 - `Mesh & getMesh ()`

Return reference to mesh instance.

 - `Mesh * getPtrMesh ()`

Return pointer to mesh.

 - void `getMomentum (Vect< real_t > &m)` const

Calculate elementwise momentum.

 - void `getInternalEnergy (Vect< real_t > &e)` const

Calculate elementwise internal energy.

 - void `getTotalEnergy (Vect< real_t > &e)` const

Return elementwise total energy.

 - void `getSoundSpeed (Vect< real_t > &s)` const

Return elementwise sound speed.

 - void `getMach (Vect< real_t > &m)` const

Return elementwise Mach number.

 - bool `setReconstruction (const Vect< real_t > &U, Vect< real_t > &LU, Vect< real_t > &RU, size_t dof)`

*Function to reconstruct by the *Muscl* method.*

 - `real_t getMinimumFaceArea ()` const

Return minimum area of faces in the mesh.

 - `real_t getMinimumElementVolume ()` const

Return minimum volume of elements in the mesh.

 - `real_t getMaximumFaceArea ()` const

Return maximum area of faces in the mesh.

 - `real_t getMaximumElementVolume ()` const

Return maximum volume of elements in the mesh.

 - `real_t getMeanFaceArea ()` const

- Return mean area of faces in the mesh.*

 - `real_t getMeanElementVolume ()` const

Return mean volume of elements in the mesh.
- `real_t getMinimumEdgeLength ()` const

Return minimum length of edges in the mesh.
- `real_t getMinimumVolumebyArea ()` const

Return minimum volume by area in the mesh.
- `real_t getMaximumEdgeLength ()` const

Return maximum length of edges in the mesh.
- `real_t getTauLim ()` const

Return value of tau lim.
- `real_t getComega ()` const

Return value of Comega.
- `void setbetalim (real_t bl)`

Assign value of beta lim.
- `Mesh & getMesh ()` const

Return reference to [Mesh](#) instance.
- `void setVerbose (int v)`

Set verbosity parameter.
- `void setMethod (const Method &s)`

Choose a flux solver.
- `void setSolidZoneCode (int c)`

Choose a code for solid zone.
- `bool getSolidZone ()` const

Return flag for presence of solid zones.
- `int getSolidZoneCode ()` const

Return code of solid zone, 0 if this one is not present.
- `void setLimiter (Limiter l)`

Choose a flux limiter.

7.47.1 Detailed Description

Class to solve the Inviscid compressible fluid flows (Euler equations) for perfect gas in 3-D.
Solution method is a second-order MUSCL Finite Volume scheme with tetrahedra

7.47.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

- FIRST_ORDER_METHOD** First Order upwind method
- MULTI_SLOPE_Q_METHOD** Multislope Q method
- MULTI_SLOPE_M_METHOD** Multislope M method
- M_FORCE_WORD** for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatability reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver
VFROE_SOLVER Finite Volume Roe solver
LF_SOLVER LF solver
RUSANOV_SOLVER Rusanov solver
HLL_SOLVER HLL solver
HLLC_SOLVER HLLC solver
MAX_SOLVER Max solver
FORCE_WORD for compatability reason, gcc may return a warning anyway

7.47.3 Constructor & Destructor Documentation

ICPG3DT (Mesh & ms)

Constructor using mesh data.

Parameters

in	ms	Mesh instance
-----------	-----------	-------------------------------

ICPG3DT (Mesh & ms, Vect< real_t > & r, Vect< real_t > & v, Vect< real_t > & p)

Constructor using mesh and initial data.

Parameters

in	ms	Mesh instance
in	r	Elementwise initial density vector (as instance of Element Vect)
in	v	Elementwise initial velocity vector (as instance of Element Vect)
in	p	Elementwise initial pressure vector (as instance of Element Vect)

7.47.4 Member Function Documentation

void setReconstruction ()

Reconstruct.

exit(3) if reconstruction failed

```
bool setReconstruction ( const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t >  
& RU, size_t dof ) [inherited]
```

Function to reconstruct by the [Muscl](#) method.

Parameters

in	U	Field to reconstruct
out	LU	Left gradient vector
out	RU	Right gradient vector
in	dof	Label of dof to reconstruct

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	v	Value of verbosity parameter
----	-----	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	s	Solver to choose
----	-----	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

Parameters

in	l	Limiter to choose
----	-----	-------------------

7.48 IOField Class Reference

Enables working with files in the XML Format.

Inherits XMLParser.

Public Types

- enum [AccessType](#)

Enumerated values for file access type.

Public Member Functions

- [IOField](#) ()
Default constructor.
- [IOField](#) (const string &file, [AccessType](#) access, bool compact=true)
Constructor using file name.
- [IOField](#) (const string &mesh_file, const string &file, [Mesh](#) &ms, [AccessType](#) access, bool compact=true)
Constructor using file name, mesh file and mesh.
- [IOField](#) (const string &file, [Mesh](#) &ms, [AccessType](#) access, bool compact=true)
Constructor using file name and mesh.
- [~IOField](#) ()
Destructor.

- void `setMeshFile` (const string &file)
Set mesh file.
- void `open` ()
Open file.
- void `open` (const string &file, `AccessType` access)
Open file.
- void `close` ()
Close file.
- void `put` (`Mesh` &ms)
Store mesh in file.
- void `put` (const `Vect`< `real_t` > &v)
Store Vect instance v in file.
- `real_t` `get` (`Vect`< `real_t` > &v)
Get Vect v instance from file.
- int `get` (`Vect`< `real_t` > &v, `real_t` t)
Get Vect v instance from file corresponding to a specific time value.
- void `saveGMSH` (string output_file, string mesh_file)
Save field vectors in a file using GMSH format.

7.48.1 Detailed Description

Enables working with files in the XML Format.

This class has methods to store vectors in files and read from files.

7.48.2 Constructor & Destructor Documentation

`IOField` (const string & file, `AccessType` access, bool compact = true)

Constructor using file name.

Parameters

in	file	File name.
in	access	Access code. This number is to be chosen among two enumerated values: <ul style="list-style-type: none"> • <code>IOField::IN</code> to read the file • <code>IOField::OUT</code> to write on it
in	compact	Flag to choose a compact storage or not [Default: true]

`IOField` (const string & mesh_file, const string & file, `Mesh` & ms, `AccessType` access, bool compact = true)

Constructor using file name, mesh file and mesh.

Parameters

in	mesh_file	File containing mesh
----	-----------	----------------------

in	<i>file</i>	File that contains field stored or to store
in	<i>ms</i>	Mesh instance
in	<i>access</i>	Access code. This number is to be chosen among two enumerated values: <ul style="list-style-type: none"> • <code>IOField::IN</code> to read the file • <code>IOField::OUT</code> to write on it
in	<i>compact</i>	Flag to choose a compact storage or not [Default: <code>true</code>]

IOField (const string & file, Mesh & ms, AccessType access, bool compact = true)

Constructor using file name and mesh.

Parameters

in	<i>file</i>	File that contains field stored or to store
in	<i>ms</i>	Mesh instance
in	<i>access</i>	Access code. This number is to be chosen among two enumerated values: <ul style="list-style-type: none"> • <code>IOField::IN</code> to read the file • <code>IOField::OUT</code> to write on it
in	<i>compact</i>	Flag to choose a compact storage or not [Default: <code>true</code>]

7.48.3 Member Function Documentation

void setMeshFile (const string & file)

Set mesh file.

Parameters

in	<i>file</i>	Mesh file
----	-------------	---------------------------

void open ()

Open file.

Case where file name has been previously given (in the constructor).

void open (const string & file, AccessType access)

Open file.

Parameters

in	<i>file</i>	File name.
in	<i>access</i>	Access code. This number is to be chosen among two enumerated values: <ul style="list-style-type: none"> • <code>IOField::IN</code> to read the file • <code>IOField::OUT</code> to write on it

```
void put ( const Vect< real_t > & v )
```

Store [Vect](#) instance `v` in file.

Parameters

in	v	Vect instance to store
----	---	--

real_t get ([Vect](#)< real_t > & v)

Get [Vect](#) v instance from file.

First time step is read from the XML file.

int get ([Vect](#)< real_t > & v, real_t t)

Get [Vect](#) v instance from file corresponding to a specific time value.

The sought vector corresponding to the time value is read from the XML file.

Parameters

in,out	v	Vector instance
in	t	Time value

void saveGMSH (string output_file, string mesh_file)

Save field vectors in a file using **GMSH** format.

This member function enables avoiding the use of `cfield`. It must be used once all field vectors have been stored in output file. It closes this file and copies its contents to a **GMSH** file.

Parameters

in	output_file	Output file name where to store using GMSH format
in	mesh_file	File containing mesh data

7.49 IPF Class Reference

To read project parameters from a file in [IPF](#) format.

Public Member Functions

- [IPF](#) ()
Default constructor.
- [IPF](#) (const string &file)
Constructor that gives the data file name.
- [IPF](#) (const string &prog, const string &file)
Constructor that reads parameters in file file and prints header information for the calling program prog. It reads parameters in [IPF](#) Format from this file.
- [~IPF](#) ()
Destructor.
- [real_t](#) getDisplay ()
Display acquired parameters.
- int getVerbose () const
*Return parameter read using keyword **Verbose**.*
- int getOutput () const
*Return parameter read using keyword **Output**.*
- int getSave () const
*Return parameter read using keyword **Save**.*

- `int getPlot () const`
*Return parameter read using keyword **Plot**.*
- `int getBC () const`
*Return parameter read using keyword **BC**.*
- `int getBF () const`
*Return parameter read using keyword **BF**.*
- `int getSF () const`
*Return parameter read using keyword **SF**.*
- `int getInit () const`
*Return parameter read using keyword **Init**.*
- `int getData () const`
*Return parameter read using keyword **Data**.*
- `size_t getNbSteps () const`
*Return parameter read using keyword **NbSteps**.*
- `size_t getNbIter () const`
*Return parameter read using keyword **NbIter**.*
- `real_t getTimeStep () const`
*Return parameter read using keyword **TimeStep**.*
- `real_t getMaxTime () const`
*Return parameter read using keyword **MaxTime**.*
- `real_t getTolerance () const`
*Return parameter read using keyword **Tolerance**.*
- `int getIntPar (size_t n=1) const`
*Return n -th parameter read using keyword **IntPar**.*
- `string getStringPar (size_t n=1) const`
*Return n -th parameter read using keyword **StringPar**.*
- `real_t getDoublePar (size_t n=1) const`
*Return n -th parameter read using keyword **DoublePar**.*
- `Point< real_t > getPointDoublePar (size_t n=1) const`
*Return n -th parameter read using keyword **PointDoublePar**.*
- `complex_t getComplexPar (size_t n=1) const`
*Return n -th parameter read using keyword **StringPar**.*
- `string getString (const string &label) const`
Return parameter corresponding to a given label, when its value is a string.
- `int getInteger (const string &label) const`
Return parameter corresponding to a given label, when its value is an integer.
- `real_t getDouble (const string &label) const`
Return parameter corresponding to a given label, when its value is a real.t.
- `complex_t getComplex (const string &label) const`
Return parameter corresponding to a given label, when its value is a complex number.
- `void get (const string &label, Vect< real_t > &a) const`
Read an array of real values, corresponding to a given label.
- `real_t getArraySize (const string &label, size_t j) const`
Return an array entry for a given label.
- `void get (const string &label, int &a) const`
Return integer parameter corresponding to a given label.

- void `get` (const string &label, `real_t` &a) const
Return real parameter corresponding to a given label.
- void `get` (const string &label, `complex_t` &a) const
Return complex parameter corresponding to a given label.
- void `get` (const string &label, string &a) const
Return string parameter corresponding to a given label.
- string `getProject` () const
Return parameter read using keyword **Project**.
- string `getDomainFile` () const
Return parameter using keyword **Mesh**.
- string `getMeshFile` (size_t i=1) const
Return *i*-th parameter read using keyword **mesh_file**.
- string `getInitFile` () const
Return parameter read using keyword **InitFile**.
- string `getRestartFile` () const
Return parameter read using keyword **RestartFile**.
- string `getBCFile` () const
Return parameter read using keyword **BCFile**.
- string `getBFFile` () const
Return parameter read using keyword **BFFile**.
- string `getSFFile` () const
Return parameter read using keyword **SFFile**.
- string `getSaveFile` () const
Return parameter read using keyword **SaveFile**.
- string `getPlotFile` (int i=1) const
Return *i*-th parameter read using keyword **PlotFile**.
- string `getPrescriptionFile` (int i=1) const
Return parameter read using keyword **DataFile**.
- string `getAuxFile` (size_t i=1) const
Return *i*-th parameter read using keyword **Auxfile**.
- string `getDensity` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for density function.
- string `getElectricConductivity` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for electric conductivity.
- string `getElectricPermittivity` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for electric permittivity.
- string `getMagneticPermeability` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for magnetic permeability.
- string `getPoissonRatio` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for Poisson ratio.
- string `getThermalConductivity` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for thermal conductivity.
- string `getRhoCp` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for density * specific heat.
- string `getViscosity` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for viscosity.
- string `getYoungModulus` () const
Return expression (to be parsed, function of *x*, *y*, *z*, *t*) for Young's modulus.

7.49.1 Detailed Description

To read project parameters from a file in [IPF](#) format.

This class can be used to acquire various parameters from a parameter file of [IPF](#) (Input Project File). The declaration of an instance of this class avoids reading data in your main program. The acquired parameters are retrieved through information members of the class. Note that all the parameters have default values

7.49.2 Constructor & Destructor Documentation

IPF (const string & file)

Constructor that gives the data file name.

It reads parameters in [IPF](#) Format from this file.

7.49.3 Member Function Documentation

int getOutput () const

Return parameter read using keyword **Output**.

This parameter can be used to control output behavior in a program.

int getSave () const

Return parameter read using keyword **Save**.

This parameter can be used to control result saving in a program (e.g. for a restarting purpose).

int getPlot () const

Return parameter read using keyword **Plot**.

This parameter can be used to control result saving for plotting in a program.

int getBC () const

Return parameter read using keyword **BC**.

This parameter can be used to set a boundary condition flag.

int getBF () const

Return parameter read using keyword **BF**.

This parameter can be used to set a body force flag.

int getSF () const

Return parameter read using keyword **SF**.

This parameter can be used to set a surface force flag.

int getInit () const

Return parameter read using keyword **Init**.

This parameter can be used to set an initial data flag.

int getData () const

Return parameter read using keyword **Data**.

This parameter can be used to set a various data flag.

size_t getNbSteps () const

Return parameter read using keyword **NbSteps**.

This parameter can be used to read a number of time steps.

size_t getNbIter () const

Return parameter read using keyword **NbIter**.

This parameter can be used to read a number of iterations.

real_t getTimeStep () const

Return parameter read using keyword **TimeStep**.

This parameter can be used to read a time step value.

real_t getMaxTime () const

Return parameter read using keyword **MaxTime**.

This parameter can be used to read a maximum time value.

real_t getTolerance () const

Return parameter read using keyword **Tolerance**.

This parameter can be used to read a tolerance value to control convergence.

int getIntPar (size_t n = 1) const

Return *n*-th parameter read using keyword **IntPar**

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for *n* is 1

string getStringPar (size_t n = 1) const

Return *n*-th parameter read using keyword **StringPar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for *n* is 1

real_t getDoublePar (size_t n = 1) const

Return *n*-th parameter read using keyword **DoublePar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for *n* is 1

Point<real_t> getPointDoublePar (size_t n = 1) const

Return *n*-th parameter read using keyword **PointDoublePar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for *n* is 1

complex_t getComplexPar (size_t n = 1) const

Return *n*-th parameter read using keyword **StringPar**.

Here we have at most 20 integer extra parameters that can be used for any purpose. Default value for *n* is 1

string getString (const string & label) const

Return parameter corresponding to a given label, when its value is a string.

Parameters

in	<i>label</i>	Label that identifies the string (read from input file). If this label is not found an error message is displayed and program stops.
-----------	--------------	--

int getInteger (const string & label) const

Return parameter corresponding to a given label, when its value is an integer.

Parameters

in	<i>label</i>	Label that identifies the integer number (read from input file). If this label is not found an error message is displayed and program stops.
-----------	--------------	--

real.t getDouble (const string & label) const

Return parameter corresponding to a given label, when its value is a real.t.

Parameters

in	<i>label</i>	Label that identifies the real number (read from input file). If this label is not found an error message is displayed and program stops.
-----------	--------------	---

complex.t getComplex (const string & label) const

Return parameter corresponding to a given label, when its value is a complex number.

Parameters

in	<i>label</i>	Label that identifies the complex number (read from input file). If this label is not found an error message is displayed and program stops.
-----------	--------------	--

void get (const string & label, Vect< real.t > & a) const

Read an array of real values, corresponding to a given label.

Parameters

in	<i>label</i>	Label that identifies the array (read from input file).
in	<i>a</i>	Vector that contain the array. The vector is properly resized before filling.

Remarks

If this label is not found an error message is displayed.

real.t getArraySize (const string & label, size.t j) const

Return an array entry for a given label.

Parameters

in	<i>label</i>	Label that identifies the array (read from input file).
in	<i>j</i>	Index of entry in the array (Starting from 1)

Remarks

If this label is not found an error message is displayed and program stops.

void get (const string & label, int & a) const

Return integer parameter corresponding to a given label.

Parameters

in	<i>label</i>	Label that identifies the integer number (read from input file).
out	<i>a</i>	Returned value. If this label is not found an error message is displayed and program stops. Note: This member function can be used instead of <code>getInteger</code>

void get (const string & label, real_t & a) const

Return real parameter corresponding to a given label.

Parameters

in	<i>label</i>	Label that identifies the real (real_t) number (read from input file).
out	<i>a</i>	Returned value. If this label is not found an error message is displayed and program stops. Note: This member function can be used instead of <code>getReal_T</code>

void get (const string & label, complex_t & a) const

Return complex parameter corresponding to a given label.

Parameters

in	<i>label</i>	Label that identifies the complex number (read from input file).
out	<i>a</i>	Returned value. If this label is not found an error message is displayed and program stops.

void get (const string & label, string & a) const

Return string parameter corresponding to a given label.

Parameters

in	<i>label</i>	Label that identifies the astring (read from input file).
out	<i>a</i>	Returned value. Note: This member function can be used instead of <code>getString</code> If this label is not found an error message is displayed and program stops. Note: This member function can be used instead of <code>getString</code>

string getProject () const

Return parameter read using keyword **Project**.

This parameter can be used to read a project's name.

string getMeshFile (size_t i = 1) const

Return i-th parameter read using keyword **mesh_file**.

Here we have at most 10 integer extra parameters that can be used for any purpose. Default value for i is 1

string getInitFile () const

Return parameter read using keyword **InitFile**.

This parameter can be used to read an initial data file name.

string getRestartFile () const

Return parameter read using keyword **RestartFile**.

This parameter can be used to read a restart file name.

string getBCFile () const

Return parameter read using keyword **BCFile**.

This parameter can be used to read a boundary condition file name.

string getBFFile () const

Return parameter read using keyword **BFFile**.

This parameter can be used to read a body force file name.

string getSFFile () const

Return parameter read using keyword **SFFile**.

This parameter can be used to read a source force file name.

string getSaveFile () const

Return parameter read using keyword **SaveFile**.

This parameter can be used to read a save file name.

string getPlotFile (int i = 1) const

Return i-th parameter read using keyword **PlotFile**.

Here we have at most 10 integer extra parameters that can be used for plot file names. Default value for i is 1

string getPrescriptionFile (int i = 1) const

Return parameter read using keyword **DataFile**.

This parameter can be used to read a [Prescription](#) file.

string getAuxFile (size_t i = 1) const

Return i-th parameter read using keyword **Auxfile**.

Here we have at most 10 integer extra parameters that can be used for any auxiliary file names. Default value for i is 1

7.50 Iter< T_ > Class Template Reference

Class to drive an iterative process.

Public Member Functions

- [Iter](#) ()
Default Constructor.
- [Iter](#) (int max_it, double toler, int verbose=0)
Constructor with iteration parameters.
- [~Iter](#) ()
Destructor.

- void `setMaxIter` (int max_it)
Set maximal number of iterations.
- void `setTolerance` (real_t toler)
Set tolerance value for convergence.
- void `setVerbose` (int v)
Set verbosity parameter.
- bool `check` (Vect< T_ > &u, const Vect< T_ > &v, int opt=2)
Check convergence.

7.50.1 Detailed Description

template<class T_>class OFELI::Iter< T_ >

Class to drive an iterative process.

This template class enables monitoring any iterative process. It simply sets default values for tolerance, maximal number of iterations and enables checking convergence using two successive iterates.

7.50.2 Constructor & Destructor Documentation

Iter ()

Default Constructor.

This constructor set default values: the maximal number of iterations is set to 100 and the tolerance to 1.e-8

Iter (int max_it, double toler, int verbose = 0)

Constructor with iteration parameters.

Parameters

in	<i>max_it</i>	Maximum number of iterations
in	<i>toler</i>	Tolerance value for convergence
in	<i>verbose</i>	Verbosity parameter [default: 0] 0: No message output, > 0: message output with increasing display.

7.50.3 Member Function Documentation

bool check (Vect< T_ > & u, const Vect< T_ > & v, int opt = 2)

Check convergence.

Parameters

in,out	<i>u</i>	Solution vector at previous iteration
in	<i>v</i>	Solution vector at current iteration
in	<i>opt</i>	Vector norm for convergence checking 1: 1-norm, 2: 2-norm, 0: Max. norm [default: 2]

Returns

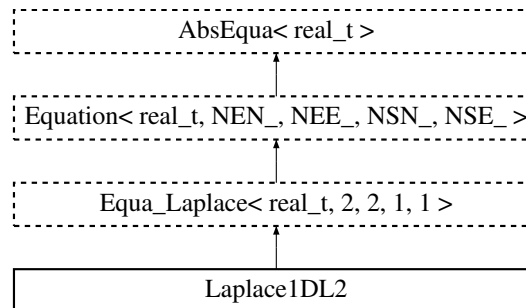
true if convergence criterion is satisfied, false if not

After checking, this function copied v into u.

7.51 Laplace1DL2 Class Reference

To build element equation for a 1-D elliptic equation using the 2-Node line element (P_1).

Inheritance diagram for Laplace1DL2:



Public Member Functions

- **Laplace1DL2** (**Element** *el)
Constructor for an element.
- **Laplace1DL2** (**Mesh** &ms, **Vect**< **real_t** > &u)
- **~Laplace1DL2** ()
Destructor.
- void **Matrix** (**real_t** coef=1.)
Add finite element matrix to left hand side.
- void **BodyRHS** (const **Vect**< **real_t** > &f)
Add Right-Hand Side Contribution.
- void **BoundaryRHS** (int n, **real_t** p)
Add Neumann contribution to Right-Hand Side.
- void **setBoundaryCondition** (**real_t** f, int lr)
Set Dirichlet boundary data.
- void **setTraction** (**real_t** f, int lr)
Set Traction data.
- int **run** ()
- void **updateBC** (const **Element** &el, const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **updateBC** (const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **DiagBC** (int dof.type=NODE.DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void **LocalNodeVector** (**Vect**< **real_t** > &b)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void **ElementNodeVectorSingleDOF** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementSideVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NSE_ > &be)

- Localize Element Vector from a Vect instance.*

 - void `ElementVector` (const `Vect< real_t >` &b, int dof_type=NODE.FIELD, int flag=0)

Localize Element Vector.
- void `SideVector` (const `Vect< real_t >` &b)

Localize Side Vector.
- void `ElementNodeCoordinates` ()

Localize coordinates of element nodes.
- void `SideNodeCoordinates` ()

Localize coordinates of side nodes.
- void `ElementAssembly` (`Matrix< real_t >` *A)

Assemble element matrix into global one.
- void `ElementAssembly` (`SkSMatrix< real_t >` &A)

Assemble element matrix into global one.
- void `ElementAssembly` (`SkMatrix< real_t >` &A)

Assemble element matrix into global one.
- void `ElementAssembly` (`SpMatrix< real_t >` &A)

Assemble element matrix into global one.
- void `ElementAssembly` (`TrMatrix< real_t >` &A)

Assemble element matrix into global one.
- void `ElementAssembly` (`Vect< real_t >` &v)

Assemble element vector into global one.
- void `DGElementAssembly` (`Matrix< real_t >` *A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkSMatrix< real_t >` &A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SkMatrix< real_t >` &A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`SpMatrix< real_t >` &A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `DGElementAssembly` (`TrMatrix< real_t >` &A)

Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void `SideAssembly` (`Matrix< real_t >` *A)

Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkSMatrix< real_t >` &A)

Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SkMatrix< real_t >` &A)

Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`SpMatrix< real_t >` &A)

Assemble side (edge or face) matrix into global one.
- void `SideAssembly` (`Vect< real_t >` &v)

Assemble side (edge or face) vector into global one.
- void `AxbAssembly` (const `Element` &el, const `Vect< real_t >` &x, `Vect< real_t >` &b)

Assemble product of element matrix by element vector into global vector.
- void `AxbAssembly` (const `Side` &sd, const `Vect< real_t >` &x, `Vect< real_t >` &b)

Assemble product of side matrix by side vector into global vector.
- size_t `getNbNodes` () const

- Return number of element nodes.*

 - `size_t getNbEq ()` const
- Return number of element equations.*

 - `real_t * A ()`
- Return element matrix as a C-array.*

 - `real_t * sA ()`
- Return side matrix as a C-array.*

 - `real_t * b ()`
- Return element right-hand side as a C-array.*

 - `real_t * sb ()`
- Return side right-hand side as a C-array.*

 - `real_t * Prev ()`
- Return element matrix as a C-array.*

 - `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
- Return element matrix as a LocalMatrix instance.*

 - `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
- Return side matrix as a LocalMatrix instance.*

 - `LocalVect< real_t, NEE_ > & Eb ()`
- Return element right-hand side as a LocalVect instance.*

 - `LocalVect< real_t, NEE_ > & Ep ()`
- Return element matrix as a C-array.*

 - `void setInitialSolution (const Vect< real_t > &u)`
- Set initial solution (previous time step)*

 - `real_t setMaterialProperty (const string &exp, const string &prop)`
- Define a material property by an algebraic expression.*

 - `void setMesh (class Mesh &m)`
- Define mesh and renumber DOFs after removing imposed ones.*

 - `Mesh & getMesh ()` const
- Return reference to Mesh instance.*

 - `LinearSolver< real_t > & getLinearSolver ()`
- Return reference to linear solver instance.*

 - `void setSolver (int ls, int pc=IDENT_PREC)`
- Choose solver for the linear system.*

 - `int solveEigenProblem (int nb_eigv, bool g=false)`
- Compute eigenvalues and eigenvectors.*

 - `real_t getEigenValue (int n)` const
- Return the n-th eigenvalue.*

 - `void getEigenvector (int n, Vect< real_t > &v)` const
- Store the eigenvector corresponding to a given eigenvalue.*

 - `class Eigen & getEigenSolver ()`
- Return reference to eigenproblem solver.*

Protected Member Functions

- `void Init (const Element *el)`
Set element arrays to zero.
- `void Init (const Side *sd)`
Set side arrays to zero.

7.51.1 Detailed Description

To build element equation for a 1-D elliptic equation using the 2-Node line element (P_1).

7.51.2 Constructor & Destructor Documentation

Laplace1DL2 (Mesh & ms, Vect< real_t > & u)

Constructor using mesh instance and solution vector

Parameters

<i>in</i>	<i>ms</i>	Mesh instance
<i>in,out</i>	<i>u</i>	Vect instance that contains, after execution of <code>run()</code> the solution

7.51.3 Member Function Documentation

void Matrix (real_t coef = 1.)

Add finite element matrix to left hand side.

Parameters

<i>in</i>	<i>coef</i>	Value to multiply by the added matrix
-----------	-------------	---------------------------------------

void BodyRHS (const Vect< real_t > & f)

Add Right-Hand Side Contribution.

Parameters

<i>in</i>	<i>f</i>	Vector containing the source given function at mesh nodes
-----------	----------	---

void BoundaryRHS (int n, real_t p)

Add Neumann contribution to Right-Hand Side.

Parameters

<i>in</i>	<i>n</i>	Parameter to select equal to 0 if the condition is at the left end of the domain and different if it is at the right of it
<i>in</i>	<i>p</i>	Value of flux to add

Note

This member function is to be called only for the first or last element

void setBoundaryCondition (real_t f, int lr)

Set Dirichlet boundary data.

Parameters

<i>in</i>	<i>f</i>	Value to assign
<i>in</i>	<i>lr</i>	Option to choose location of the value (-1: Left end, 1: Right end)

void setTraction (real_t f, int lr)

Set Traction data.

Parameters

in	<i>f</i>	Value of traction (Neumann boundary condition)
in	<i>lr</i>	Option to choose location of the traction (-1: Left end, 1: Right end)

int run ()

Run solution procedure This function is to be called when the constructor **Laplace1DL2(mesh,u)** is used.

Returns

return code for the solution of the linear system

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by *_theElement*

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE.FIELD, DOFs are supported by nodes [Default] • ELEMENT.FIELD, DOFs are supported by elements • SIDE.FIELD, DOFs are supported by sides
in	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
-----------	-----------------	--

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real_t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number dof is handled in the system The resulting local vector can be accessed by attribute ePrev.

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer `_theElement`

void SideVector (const Vect< real_t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute ePrev.
----	----------	--

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an <code>SkSMatrix</code> instance
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)

out	<i>b</i>	Global vector to add (Vect instance)
-----	----------	--------------------------------------

void AxbAssembly (const Side & sd, const Vect< real.t > & x, Vect< real.t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real.t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

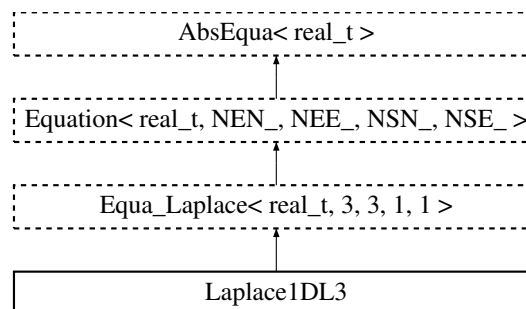
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.52 Laplace1DL3 Class Reference

To build element equation for the 1-D elliptic equation using the 3-Node line (P_2).

Inheritance diagram for Laplace1DL3:



Public Member Functions

- **Laplace1DL3** (**Mesh** &ms, **Vect**< **real_t** > &u)
- **Laplace1DL3** (**Element** *el)
Constructor for an element.
- **~Laplace1DL3** ()
Destructor.
- void **Matrix** (**real_t** coef=1.)
Add finite element matrix to left hand side.
- void **BodyRHS** (const **Vect**< **double** > &f)
Add Right-hand side contribution.
- void **BoundaryRHS** (int n, **real_t** p)
Add Neumann contribution to Right-Hand Side.
- void **setTraction** (**real_t** f, int lr)
Set Traction data.
- int **run** ()
- void **updateBC** (const **Element** &el, const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **updateBC** (const **Vect**< **real_t** > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **DiagBC** (int dof.type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void **LocalNodeVector** (**Vect**< **real_t** > &b)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const **Vect**< **real_t** > &b, **LocalVect**< **real_t**, NEN_ > &be, int dof)

- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVectorSingleDOF` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementSideVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NSE_ >` &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementVector` (const `Vect< real_t >` &b, int dof_type=NODE.FIELD, int flag=0)
- Localize Element Vector.*

 - void `SideVector` (const `Vect< real_t >` &b)
- Localize Side Vector.*

 - void `ElementNodeCoordinates` ()
- Localize coordinates of element nodes.*

 - void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*

 - void `ElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`Vect< real_t >` &v)
- Assemble element vector into global one.*

 - void `DGElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `SideAssembly` (`Matrix< real_t >` *A)
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly` (`SkSMatrix< real_t >` &A)
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly` (`SkMatrix< real_t >` &A)
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly` (`SpMatrix< real_t >` &A)
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly` (`Vect< real_t >` &v)
- Assemble side (edge or face) vector into global one.*

 - void `AxbAssembly` (const `Element` &el, const `Vect< real_t >` &x, `Vect< real_t >` &b)

- Assemble product of element matrix by element vector into global vector.*

 - `void AxbAssembly (const Side &sd, const Vect< real_t > &x, Vect< real_t > &b)`

Assemble product of side matrix by side vector into global vector.
- `size_t getNbNodes () const`

Return number of element nodes.
- `size_t getNbEq () const`

Return number of element equations.
- `real_t * A ()`

Return element matrix as a C-array.
- `real_t * sA ()`

Return side matrix as a C-array.
- `real_t * b ()`

Return element right-hand side as a C-array.
- `real_t * sb ()`

Return side right-hand side as a C-array.
- `real_t * Prev ()`

Return element matrix as a C-array.
- `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`

Return element matrix as a LocalMatrix instance.
- `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`

Return side matrix as a LocalMatrix instance.
- `LocalVect< real_t, NEE_ > & Eb ()`

Return element right-hand side as a LocalVect instance.
- `LocalVect< real_t, NEE_ > & Ep ()`

Return element matrix as a C-array.
- `void setInitialSolution (const Vect< real_t > &u)`

Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`

Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`

Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`

Return reference to Mesh instance.
- `LinearSolver< real_t > & getLinearSolver ()`

Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`

Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`

Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`

Return the n-th eigenvalue.
- `void getEigenvector (int n, Vect< real_t > &v) const`

Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`

Return reference to eigenproblem solver.

Protected Member Functions

- void `Init` (const `Element` *el)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.52.1 Detailed Description

To build element equation for the 1-D elliptic equation using the 3-Node line (P_2).

7.52.2 Constructor & Destructor Documentation

Laplace1DL3 (`Mesh` & ms, `Vect`< `real_t` > & u)

Constructor using mesh instance and solution vector

Parameters

in	ms	<code>Mesh</code> instance
in,out	u	<code>Vect</code> instance that contains, after execution of <code>run()</code> the solution

7.52.3 Member Function Documentation

void Matrix (`real_t` coef = 1.)

Add finite element matrix to left hand side.

Parameters

in	coef	Value to multiply by the added matrix
----	------	---------------------------------------

void BodyRHS (`const Vect`< `double` > & f)

Add Right-hand side contribution.

Parameters

in	f	Vector of right-hand side of the Poisson equation at nodes
----	---	--

void BoundaryRHS (`int` n, `real_t` p)

Add Neumann contribution to Right-Hand Side.

Parameters

in	n	Parameter to select equal to 0 if the condition is at the left end of the domain and different if it is at the right of it
in	p	Value of flux to add

Note

This member function is to be invoked only for the first or last element

void setTraction (`real_t` f, `int` lr)

Set Traction data.

Parameters

in	<i>f</i>	Value of traction (Neumann boundary condition)
in	<i>lr</i>	Option to choose location of the traction (-1: Left end, 1: Right end)

int run ()

Run solution procedure This function is to be called when the constructor **Laplace1DL2(mesh,u)** is used.

Returns

return code for the solution of the linear system

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by *_theElement*

void DiagBC (int dof_type = NODE_DOF, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
-----------	-----------------	--

in	dof	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. dof is handled in the system
-----------	------------	--

void LocalNodeVector (Vect< real.t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute ePrev. This member function is to be used if a constructor with Element was invoked.
-----------	----------	--

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real_t > & b, LocalVect< real_t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real_t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system The resulting local vector can be accessed by attribute <i>ePrev</i> .

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real_t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides The resulting local vector can be accessed by attribute <i>ePrev</i> .
----	----------	--

Remarks

This member function is to be used if a constructor with Side was invoked. It uses the Side pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Side pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Global matrix stored as an <code>SkSMatrix</code> instance
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>in</code>	<code>A</code>	Global matrix stored as an <code>SkMatrix</code> instance
-----------------	----------------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void DGELEMENTAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void DGELEMENTAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable theElement

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
-----------	-----------	---

in	pc	Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values: <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner
-----------	-----------	---

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real.t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

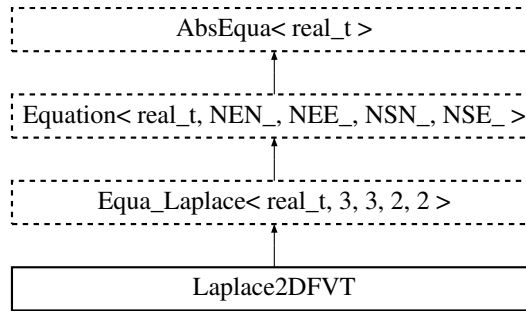
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.53 Laplace2DFVT Class Reference

To build and solve the Laplace equation using a standard Finite Volume method.

Inheritance diagram for Laplace2DFVT:



Public Member Functions

- `Laplace2DFVT (Mesh &ms, Vect< real_t > &b, Vect< real_t > &u)`
Standard constructor.
- `Laplace2DFVT (Mesh &ms, SpMatrix< real_t > &A, Vect< real_t > &b)`
Standard constructor.
- `~Laplace2DFVT ()`
Destructor.
- `int checkDelaunay (int verb=0)`
Check whether triangles are Delaunay ones.
- `void build (const Vect< real_t > &f)`
Build the linear system of equations.
- `int run (const Vect< real_t > &f)`
Build and solve the linear system of equations.
- `void LHS (const Element *e1, const Element *e2)`
Calculate left-hand side.
- `void RHS (const Vect< real_t > &f)`
Add right-hand side Contribution.
- `int run ()`
Solve the equation.
- `void updateBC (const Element &el, const Vect< real_t > &bc)`
Update Right-Hand side by taking into account essential boundary conditions.
- `void updateBC (const Vect< real_t > &bc)`
Update Right-Hand side by taking into account essential boundary conditions.
- `void DiagBC (int dof.type=NODE_DOF, int dof=0)`
Update element matrix to impose bc by diagonalization technique.
- `void LocalNodeVector (Vect< real_t > &b)`
Localize Element Vector from a Vect instance.
- `void ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEE_ > &be)`
Localize Element Vector from a Vect instance.
- `void ElementNodeVector (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be, int dof)`
Localize Element Vector from a Vect instance.
- `void ElementNodeVectorSingleDOF (const Vect< real_t > &b, LocalVect< real_t, NEN_ > &be)`
Localize Element Vector from a Vect instance.
- `void ElementSideVector (const Vect< real_t > &b, LocalVect< real_t, NSE_ > &be)`
Localize Element Vector from a Vect instance.
- `void ElementVector (const Vect< real_t > &b, int dof.type=NODE_FIELD, int flag=0)`

- Localize Element Vector.*
 - void `SideVector` (const `Vect< real_t >` &b)
 - Localize Side Vector.*
 - void `ElementNodeCoordinates` ()
 - Localize coordinates of element nodes.*
 - void `SideNodeCoordinates` ()
 - Localize coordinates of side nodes.*
 - void `ElementAssembly` (`Matrix< real_t >` *A)
 - Assemble element matrix into global one.*
 - void `ElementAssembly` (`SkSMatrix< real_t >` &A)
 - Assemble element matrix into global one.*
 - void `ElementAssembly` (`SkMatrix< real_t >` &A)
 - Assemble element matrix into global one.*
 - void `ElementAssembly` (`SpMatrix< real_t >` &A)
 - Assemble element matrix into global one.*
 - void `ElementAssembly` (`TrMatrix< real_t >` &A)
 - Assemble element matrix into global one.*
 - void `ElementAssembly` (`Vect< real_t >` &v)
 - Assemble element vector into global one.*
 - void `DGElementAssembly` (`Matrix< real_t >` *A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
 - void `DGElementAssembly` (`SkSMatrix< real_t >` &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
 - void `DGElementAssembly` (`SkMatrix< real_t >` &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
 - void `DGElementAssembly` (`SpMatrix< real_t >` &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
 - void `DGElementAssembly` (`TrMatrix< real_t >` &A)
 - Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
 - void `SideAssembly` (`Matrix< real_t >` *A)
 - Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`SkSMatrix< real_t >` &A)
 - Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`SkMatrix< real_t >` &A)
 - Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`SpMatrix< real_t >` &A)
 - Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`Vect< real_t >` &v)
 - Assemble side (edge or face) vector into global one.*
 - void `AxbAssembly` (const `Element` &el, const `Vect< real_t >` &x, `Vect< real_t >` &b)
 - Assemble product of element matrix by element vector into global vector.*
 - void `AxbAssembly` (const `Side` &sd, const `Vect< real_t >` &x, `Vect< real_t >` &b)
 - Assemble product of side matrix by side vector into global vector.*
 - size_t `getNbNodes` () const
 - Return number of element nodes.*
 - size_t `getNbEq` () const

- Return number of element equations.*

 - `real_t * A ()`

Return element matrix as a C-array.
- `real_t * sA ()`

Return side matrix as a C-array.
- `real_t * b ()`

Return element right-hand side as a C-array.
- `real_t * sb ()`

Return side right-hand side as a C-array.
- `real_t * Prev ()`

Return element matrix as a C-array.
- `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`

Return element matrix as a LocalMatrix instance.
- `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`

Return side matrix as a LocalMatrix instance.
- `LocalVect< real_t, NEE_ > & Eb ()`

Return element right-hand side as a LocalVect instance.
- `LocalVect< real_t, NEE_ > & Ep ()`

Return element matrix as a C-array.
- `void setInitialSolution (const Vect< real_t > &u)`

Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`

Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`

Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`

Return reference to Mesh instance.
- `LinearSolver< real_t > & getLinearSolver ()`

Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`

Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`

Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`

Return the n-th eigenvalue.
- `void getEigenVector (int n, Vect< real_t > &v) const`

Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`

Return reference to eigenproblem solver.

Protected Member Functions

- `void Init (const Element *el)`

Set element arrays to zero.
- `void Init (const Side *sd)`

Set side arrays to zero.

7.53.1 Detailed Description

To build and solve the Laplace equation using a standard Finite Volume method.

7.53.2 Constructor & Destructor Documentation

Laplace2DFVT (Mesh & ms, Vect< real_t > & b, Vect< real_t > & u)

Standard constructor.

Parameters

in	<i>ms</i>	Mesh instance
in	<i>b</i>	Vect instance that contains Right-hand side
in	<i>u</i>	Vect instance that contains solution

Laplace2DFVT ([Mesh](#) & *ms*, [SpMatrix](#)< [real_t](#) > & *A*, [Vect](#)< [real_t](#) > & *b*)

Standard constructor.

Parameters

in	<i>ms</i>	Mesh instance. The mesh must have been assigned the attribute <code>ELEMENT_DOF</code> to say that unknowns are supported by elements.
in	<i>A</i>	Problem matrix to be stored in sparse format (class SpMatrix)
in	<i>b</i>	Vect instance that contains Right-hand side

7.53.3 Member Function Documentation

int checkDelaunay (int *verb* = 0)

Check whether triangles are Delaunay ones.

Parameters

in	<i>verb</i>	Output (>0) or not (0) list of failing elements
----	-------------	---

Returns

ret Number of Non Delaunay triangles

void updateBC (const [Element](#) & *el*, const [Vect](#)< [real_t](#) > & *bc*) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const [Vect](#)< [real_t](#) > & *bc*) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
----	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int *dof_type* = `NODE_DOF`, int *dof* = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	----------	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector


```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Element` pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<i>A</i>	Global matrix stored as an <code>SkSMMatrix</code> instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The side pointer is given by the global variable theSide

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The side pointer is given by the global variable theSide

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	el	Reference to Element instance
in	x	Global vector to multiply by (Vect instance)
out	b	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

<code>in</code>	<code>sd</code>	Reference to Side instance
<code>in</code>	<code>x</code>	Global vector to multiply by (Vect instance)
<code>out</code>	<code>b</code>	Global vector (Vect instance)

`real_t setMaterialProperty (const string & exp, const string & prop)` [inherited]

Define a material property by an algebraic expression.

Parameters

<code>in</code>	<code>exp</code>	Algebraic expression
<code>in</code>	<code>prop</code>	Property name

Returns

Return value in expression evaluation:

- `=0`, Normal evaluation
- `!=0`, An error message is displayed

`Mesh& getMesh () const` [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

`void setSolver (int ls, int pc = IDENT_PREC)` [inherited]

Choose solver for the linear system.

Parameters

in	<i>ls</i>	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	<i>pc</i>	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	<i>nb_eigv</i>	Number of eigenvalues to compute
in	<i>g</i>	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenvector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

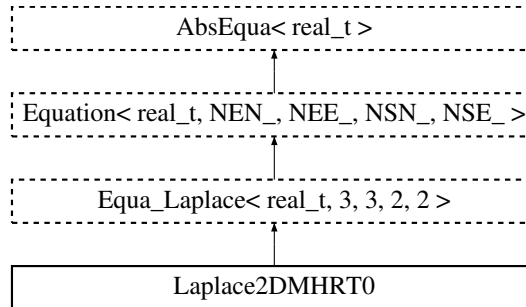
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.54 Laplace2DMHRT0 Class Reference

To build element equation for the 2-D elliptic equation using the Mixed Hybrid finite element at lowest degree (Raviart-Thomas RT_0).

Inheritance diagram for Laplace2DMHRT0:



Public Member Functions

- [Laplace2DMHRT0 \(\)](#)
Default Constructor.
- [Laplace2DMHRT0 \(Mesh &ms, SpMatrix< real_t > &A, Vect< real_t > &b\)](#)
Constructor with problem data.
- [~Laplace2DMHRT0 \(\)](#)
Destructor.
- void [setDiffusivity](#) (const [LocalMatrix](#)< real_t, 2, 2 > &K)
Define Diffusivity (or permeability) matrix.
- void [build](#) ()
Build global matrix and right-hand side.
- void [Post](#) (const [Vect](#)< real_t > &lambda, const [Vect](#)< real_t > &f, [Vect](#)< real_t > &v, [Vect](#)< [Point](#)< real_t > > &p, [Vect](#)< real_t > &u)
Perform post calculations.
- int [solve](#) ([Vect](#)< real_t > &u)
Solve the linear system of equations using the Conjugate Gradient iterative method.
- int [run](#) ()
Solve the equation.
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< real_t > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< real_t > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void [LocalNodeVector](#) ([Vect](#)< real_t > &b)

- Localize Element Vector from a Vect instance.*
- void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEE_ >` &be)
- Localize Element Vector from a Vect instance.*
- void `ElementNodeVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be, int dof)
- Localize Element Vector from a Vect instance.*
- void `ElementNodeVectorSingleDOF` (const `Vect< real_t >` &b, `LocalVect< real_t, NEN_ >` &be)
- Localize Element Vector from a Vect instance.*
- void `ElementSideVector` (const `Vect< real_t >` &b, `LocalVect< real_t, NSE_ >` &be)
- Localize Element Vector from a Vect instance.*
- void `ElementVector` (const `Vect< real_t >` &b, int dof_type=NODE.FIELD, int flag=0)
- Localize Element Vector.*
- void `SideVector` (const `Vect< real_t >` &b)
- Localize Side Vector.*
- void `ElementNodeCoordinates` ()
- Localize coordinates of element nodes.*
- void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*
- void `ElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one.*
- void `ElementAssembly` (`Vect< real_t >` &v)
- Assemble element vector into global one.*
- void `DGElementAssembly` (`Matrix< real_t >` *A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkSMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SkMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`SpMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `DGElementAssembly` (`TrMatrix< real_t >` &A)
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*
- void `SideAssembly` (`Matrix< real_t >` *A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkSMatrix< real_t >` &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SkMatrix< real_t >` &A)
- Assemble side (edge or face) matrix into global one.*
- void `SideAssembly` (`SpMatrix< real_t >` &A)

- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`Vect< real_t > &v`)
- Assemble side (edge or face) vector into global one.*
 - void `AxbAssembly` (const `Element &el`, const `Vect< real_t > &x`, `Vect< real_t > &b`)
- Assemble product of element matrix by element vector into global vector.*
 - void `AxbAssembly` (const `Side &sd`, const `Vect< real_t > &x`, `Vect< real_t > &b`)
- Assemble product of side matrix by side vector into global vector.*
 - size_t `getNbNodes` () const
- Return number of element nodes.*
 - size_t `getNbEq` () const
- Return number of element equations.*
 - `real_t * A` ()
- Return element matrix as a C-array.*
 - `real_t * sA` ()
- Return side matrix as a C-array.*
 - `real_t * b` ()
- Return element right-hand side as a C-array.*
 - `real_t * sb` ()
- Return side right-hand side as a C-array.*
 - `real_t * Prev` ()
- Return element matrix as a C-array.*
 - `LocalMatrix< real_t, NEE_, NEE_ > & EA` ()
- Return element matrix as a LocalMatrix instance.*
 - `LocalMatrix< real_t, NSE_, NSE_ > & SA` ()
- Return side matrix as a LocalMatrix instance.*
 - `LocalVect< real_t, NEE_ > & Eb` ()
- Return element right-hand side as a LocalVect instance.*
 - `LocalVect< real_t, NEE_ > & Ep` ()
- Return element matrix as a C-array.*
 - void `setInitialSolution` (const `Vect< real_t > &u`)
- Set initial solution (previous time step)*
 - `real_t setMaterialProperty` (const string &exp, const string &prop)
- Define a material property by an algebraic expression.*
 - void `setMesh` (class `Mesh &m`)
- Define mesh and renumber DOFs after removing imposed ones.*
 - `Mesh & getMesh` () const
- Return reference to Mesh instance.*
 - `LinearSolver< real_t > & getLinearSolver` ()
- Return reference to linear solver instance.*
 - void `setSolver` (int ls, int pc=IDENT_PREC)
- Choose solver for the linear system.*
 - int `solveEigenProblem` (int nb_eigv, bool g=false)
- Compute eigenvalues and eigenvectors.*
 - `real_t getEigenValue` (int n) const
- Return the n-th eigenvalue.*
 - void `getEigenvector` (int n, `Vect< real_t > &v`) const
- Store the eigenvector corresponding to a given eigenvalue.*
 - class `Eigen & getEigenSolver` ()
- Return reference to eigenproblem solver.*

Protected Member Functions

- void `Init` (const `Element` *el)
Set element arrays to zero.
- void `Init` (const `Side` *sd)
Set side arrays to zero.

7.54.1 Detailed Description

To build element equation for the 2-D elliptic equation using the Mixed Hybrid finite element at lowest degree (Raviart-Thomas RT_0).

7.54.2 Constructor & Destructor Documentation

`Laplace2DMHRT0` ()

Default Constructor.

Constructs an empty equation.

`Laplace2DMHRT0` (`Mesh` & ms, `SpMatrix`< `real_t` > & A, `Vect`< `real_t` > & b)

Constructor with problem data.

Parameters

in	<i>ms</i>	<code>Mesh</code> instance
in	<i>A</i>	Problem matrix in Sparse format. This matrix must be zeroed before calling the constructor
in	<i>b</i>	Problem right-hand side

7.54.3 Member Function Documentation

`void setDiffusivity` (const `LocalMatrix`< `real_t`, 2, 2 > & K)

Define Diffusivity (or permeability) matrix.

By default (if this function is not called) the identity matrix (Laplace equation) is used.

Parameters

in	<i>K</i>	Diffusivity matrix as <code>LocalMatrix</code> instance. Must be symmetric positive definite
----	----------	--

`void build` () [virtual]

Build global matrix and right-hand side.

The problem matrix and right-hand side are the ones used in the constructor. They are updated in this member function.

Reimplemented from `Equa_Laplace`< `real_t`, 3, 3, 2, 2 >.

`void Post` (const `Vect`< `real_t` > & lambda, const `Vect`< `real_t` > & f, `Vect`< `real_t` > & v, `Vect`< `Point`< `real_t` > > & p, `Vect`< `real_t` > & u)

Perform post calculations.

Parameters

in	<i>lambda</i>	Solution (Lagrange multiplier) calculated at edges
in	<i>f</i>	Vector containing the right-hand side of the Laplace equation
in	<i>v</i>	Vector containing solution at mesh nodes
in	<i>p</i>	Vector containing gradient at elements
in	<i>u</i>	Vector containing solution at elements

int solve (Vect< real.t > & u)

Solve the linear system of equations using the Conjugate Gradient iterative method.

The matrix is preconditioned by an ILU method.

Parameters

out	<i>u</i>	Vector containing the solution at all sides (Sides where boundary conditions are prescribed are included).
------------	----------	--

Returns

Number of performed iterations in the CG method. Note that the maximal number is 1000 and the tolerance is 1.e-8

void updateBC (const Element & el, const Vect< real.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	dof_type	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	dof	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
----	---	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	b	Global vector to be localized.
out	be	Local vector, the length of which is the total number of element equations.
in	dof	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

```
void ElementNodeVectorSingleDOF ( const Vect< real_t > & b, LocalVect< real_t , NEN_  
> & be ) [inherited]
```

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0)
[inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • NODE_FIELD, DOFs are supported by nodes [Default] • ELEMENT_FIELD, DOFs are supported by elements • SIDE_FIELD, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF number <i>dof</i> is handled in the system <p>The resulting local vector can be accessed by attribute <i>ePrev</i>.</p>

Remarks

This member function is to be used if a constructor with Element was invoked. It uses the Element pointer *_theElement*

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

<code>in</code>	<code>b</code>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------------	----------------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real_t>](#)

Remarks

This member function uses the `Element` pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Pointer to global matrix (abstract class: can be any of classes <code>SkSMatrix</code> , <code>SkMatrix</code> , <code>SpMatrix</code>)
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>A</code>	Global matrix stored as an <code>SkSMatrix</code> instance
----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>in</code>	<code>A</code>	Global matrix stored as an SkMatrix instance
-----------------	----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>in</code>	<code>A</code>	Global matrix stored as an SpMatrix instance
-----------------	----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

<code>in</code>	<code>A</code>	Global matrix stored as an TrMatrix instance
-----------------	----------------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

<code>in</code>	<code>v</code>	Global vector (Vect instance)
-----------------	----------------	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	----------	---

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
-----------	----------	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	v	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function getEigen was called with an argument nb_eigv greater or equal to n. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

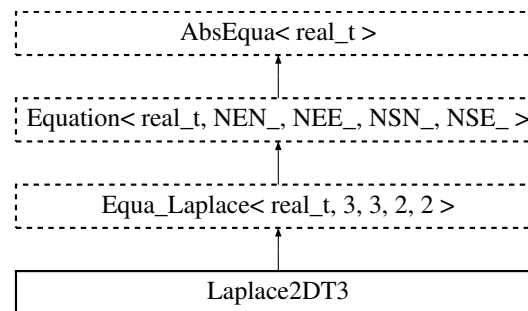
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.55 Laplace2DT3 Class Reference

To build element equation for the Laplace equation using the 2-D triangle element (P_1).

Inheritance diagram for Laplace2DT3:



Public Member Functions

- [Laplace2DT3](#) ([Mesh](#) &ms)
Constructor with mesh.
- [Laplace2DT3](#) ([Mesh](#) &ms, [SpMatrix](#)< [real_t](#) > &A, [Vect](#)< [real_t](#) > &b)
Constructor with problem data.
- [Laplace2DT3](#) ([Mesh](#) &ms, [Vect](#)< [real_t](#) > &b)
Constructor using mesh and solution vector.
- [Laplace2DT3](#) ([Element](#) *el)
Constructor for an element.
- [Laplace2DT3](#) ([Side](#) *sd)
Constructor for a side.
- [~Laplace2DT3](#) ()
Destructor.
- void [LHS](#) ([real_t](#) coef=1.)
Add finite element matrix to left-hand side.
- void [BodyRHS](#) (const [Vect](#)< [real_t](#) > &f)
Add body source term to right-hand side.
- void [BoundaryRHS](#) (const [Vect](#)< [real_t](#) > &h)
Add boundary source term to right-hand side.
- void [setDiffusivity](#) (const [LocalMatrix](#)< [real_t](#), 2, 2 > &K)
Define Diffusivity (or permeability) matrix.
- void [setSource](#) (const [Vect](#)< [real_t](#) > &f)
Define Source right-hand side of the equation.
- void [build](#) ()
Build global matrix and right-hand side.
- void [buildEigen](#) ([SkSMatrix](#)< [real_t](#) > &K, [SkSMatrix](#)< [real_t](#) > &M)

- Build global stiffness and mass matrices for the eigen system.*

 - void `buildEigen` (`SkSMatrix`< `real_t` > &K, `Vect`< `real_t` > &M)
- Build global stiffness and mass matrices for the eigen system.*

 - void `Post` (const `Vect`< `real_t` > &u, `Vect`< `Point`< `real_t` > > &p)
- Perform post calculations.*

 - int `solve` (`Vect`< `real_t` > &u)
- Solve the linear system of equations using the Conjugate Gradient iterative method.*

 - void `Axb` (const `Vect`< `real_t` > &x, `Vect`< `real_t` > &b)
- Compute the product of the stiffness matrix by a given vector.*

 - int `run` ()
- Solve the equation.*

 - void `updateBC` (const `Element` &el, const `Vect`< `real_t` > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `updateBC` (const `Vect`< `real_t` > &bc)
- Update Right-Hand side by taking into account essential boundary conditions.*

 - void `DiagBC` (int dof_type=NODE_DOF, int dof=0)
- Update element matrix to impose bc by diagonalization technique.*

 - void `LocalNodeVector` (`Vect`< `real_t` > &b)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEE_ > &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be, int dof)
- Localize Element Vector from a Vect instance.*

 - void `ElementNodeVectorSingleDOF` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NEN_ > &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementSideVector` (const `Vect`< `real_t` > &b, `LocalVect`< `real_t`, NSE_ > &be)
- Localize Element Vector from a Vect instance.*

 - void `ElementVector` (const `Vect`< `real_t` > &b, int dof_type=NODE_FIELD, int flag=0)
- Localize Element Vector.*

 - void `SideVector` (const `Vect`< `real_t` > &b)
- Localize Side Vector.*

 - void `ElementNodeCoordinates` ()
- Localize coordinates of element nodes.*

 - void `SideNodeCoordinates` ()
- Localize coordinates of side nodes.*

 - void `ElementAssembly` (`Matrix`< `real_t` > *A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkSMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SkMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`SpMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`TrMatrix`< `real_t` > &A)
- Assemble element matrix into global one.*

 - void `ElementAssembly` (`Vect`< `real_t` > &v)

- Assemble element vector into global one.*

 - void `DGElementAssembly (Matrix< real_t > *A)`
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly (SkSMatrix< real_t > &A)`
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly (SkMatrix< real_t > &A)`
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly (SpMatrix< real_t > &A)`
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `DGElementAssembly (TrMatrix< real_t > &A)`
- Assemble element matrix into global one for the Discontinuous Galerkin approximation.*

 - void `SideAssembly (Matrix< real_t > *A)`
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly (SkSMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly (SkMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly (SpMatrix< real_t > &A)`
- Assemble side (edge or face) matrix into global one.*

 - void `SideAssembly (Vect< real_t > &v)`
- Assemble side (edge or face) vector into global one.*

 - void `AxbAssembly (const Element &el, const Vect< real_t > &x, Vect< real_t > &b)`
- Assemble product of element matrix by element vector into global vector.*

 - void `AxbAssembly (const Side &sd, const Vect< real_t > &x, Vect< real_t > &b)`
- Assemble product of side matrix by side vector into global vector.*

 - size_t `getNbNodes ()` const
- Return number of element nodes.*

 - size_t `getNbEq ()` const
- Return number of element equations.*

 - `real_t * A ()`
- Return element matrix as a C-array.*

 - `real_t * sA ()`
- Return side matrix as a C-array.*

 - `real_t * b ()`
- Return element right-hand side as a C-array.*

 - `real_t * sb ()`
- Return side right-hand side as a C-array.*

 - `real_t * Prev ()`
- Return element matrix as a C-array.*

 - `LocalMatrix< real_t, NEE_, NEE_ > & EA ()`
- Return element matrix as a LocalMatrix instance.*

 - `LocalMatrix< real_t, NSE_, NSE_ > & SA ()`
- Return side matrix as a LocalMatrix instance.*

 - `LocalVect< real_t, NEE_ > & Eb ()`
- Return element right-hand side as a LocalVect instance.*

 - `LocalVect< real_t, NEE_ > & Ep ()`

- *Return element matrix as a C-array.*
- void `setInitialSolution` (const `Vect< real_t >` &u)
 - *Set initial solution (previous time step)*
- `real_t setMaterialProperty` (const string &exp, const string &prop)
 - *Define a material property by an algebraic expression.*
- void `setMesh` (class `Mesh` &m)
 - *Define mesh and renumber DOFs after removing imposed ones.*
- `Mesh` & `getMesh` () const
 - *Return reference to Mesh instance.*
- `LinearSolver< real_t >` & `getLinearSolver` ()
 - *Return reference to linear solver instance.*
- void `setSolver` (int ls, int pc=`IDENT_PREC`)
 - *Choose solver for the linear system.*
- int `solveEigenProblem` (int nb_eigv, bool g=false)
 - *Compute eigenvalues and eigenvectors.*
- `real_t getEigenValue` (int n) const
 - *Return the n-th eigenvalue.*
- void `getEigenvector` (int n, `Vect< real_t >` &v) const
 - *Store the eigenvector corresponding to a given eigenvalue.*
- class `Eigen` & `getEigenSolver` ()
 - *Return reference to eigenproblem solver.*

Protected Member Functions

- void `Init` (const `Element` *el)
 - *Set element arrays to zero.*
- void `Init` (const `Side` *sd)
 - *Set side arrays to zero.*

7.55.1 Detailed Description

To build element equation for the Laplace equation using the 2-D triangle element (P_1).

7.55.2 Constructor & Destructor Documentation

Laplace2DT3 (`Mesh` & ms)

Constructor with mesh.
Parameters

<code>in</code>	<code>ms</code>	<code>Mesh</code> instance
-----------------	-----------------	----------------------------

Laplace2DT3 (`Mesh` & ms, `SpMatrix< real_t >` & A, `Vect< real_t >` & b)

Constructor with problem data.
Parameters

<code>in</code>	<code>ms</code>	Mesh instance
<code>in</code>	<code>A</code>	Problem matrix in Sparse format. This matrix must be zeroed before calling the constructor
<code>in</code>	<code>b</code>	Problem right-hand side

Laplace2DT3 (`Mesh & ms, Vect< real.t > & b`)

Constructor using mesh and solution vector.

Parameters

<code>in</code>	<code>ms</code>	Mesh instance
<code>in</code>	<code>b</code>	Problem right-hand side

7.55.3 Member Function Documentation

void LHS (`real.t coef = 1.`)

Add finite element matrix to left-hand side.

Parameters

<code>in</code>	<code>coef</code>	Value to multiply by the added matrix
-----------------	-------------------	---------------------------------------

void BodyRHS (`const Vect< real.t > & f`)

Add body source term to right-hand side.

Parameters

<code>in</code>	<code>f</code>	Vector containing the source given function at mesh nodes
-----------------	----------------	---

void BoundaryRHS (`const Vect< real.t > & h`)

Add boundary source term to right-hand side.

Parameters

<code>in</code>	<code>h</code>	Vector containing the source given function at mesh nodes
-----------------	----------------	---

void setDiffusivity (`const LocalMatrix< real.t, 2, 2 > & K`)

Define Diffusivity (or permeability) matrix.

By default (if this function is not called) the identity matrix (Laplace equation) is used.

Parameters

<code>in</code>	<code>K</code>	Diffusivity matrix as LocalMatrix instance. Must be symmetric positive definite
-----------------	----------------	---

void setSource (`const Vect< real.t > & f`)

Define Source right-hand side of the equation.

Parameters

f	[in] Vector containing source values at nodes
-----	---

void build () [virtual]

Build global matrix and right-hand side.

The problem matrix and right-hand side are the ones used in the constructor. They are updated in this member function.

Reimplemented from [Equa_Laplace< real_t, 3, 3, 2, 2 >](#).

void buildEigen (SkSMatrix< real_t > & K, SkSMatrix< real_t > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is consistent

Parameters

in	K	Stiffness matrix
in	M	Consistent mass matrix

void buildEigen (SkSMatrix< real_t > & K, Vect< real_t > & M)

Build global stiffness and mass matrices for the eigen system.

Case where the mass matrix is lumped

Parameters

in	K	Stiffness matrix
in	M	Vector containing diagonal mass matrix

void Post (const Vect< real_t > & u, Vect< Point< real_t > > & p)

Perform post calculations.

Parameters

in	u	Solution at nodes
out	p	Vector containing gradient at elements

int solve (Vect< real_t > & u)

Solve the linear system of equations using the Conjugate Gradient iterative method.

The matrix is preconditioned by an ILU method.

Parameters

in	u	Vector containing the solution at all sides (Sides where boundary conditions are prescribed are included).
----	-----	--

Returns

Number of performed iterations in the CG method. Note that the maximal number is 1000 and the tolerance is $1.e-8$

void Axb (const Vect< real_t > & x, Vect< real_t > & b)

Compute the product of the stiffness matrix by a given vector.

Parameters

<code>in</code>	<code>x</code>	Vector by which the matrix is multiplied
<code>in</code>	<code>b</code>	Product vector

void updateBC (const Element & el, const Vect< real.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<code>in</code>	<code>el</code>	Reference to current element instance
<code>in</code>	<code>bc</code>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real.t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<code>in</code>	<code>bc</code>	Vector that contains imposed values at all DOFs
-----------------	-----------------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE.DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<code>in</code>	<code>dof_type</code>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
<code>in</code>	<code>dof</code>	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< real.t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<code>in</code>	<code>b</code>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
-----------------	----------------	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.
<i>in</i>	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only the degree of freedom is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF number <code>dof</code> is handled in the system The resulting local vector can be accessed by attribute <code>ePrev</code> .

Remarks

This member function is to be used if a constructor with `Element` was invoked. It uses the `Element` pointer `_theElement`

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGELEMENTAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real_t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function `getEigen` was called with an argument `nb_eigv` greater or equal to `n`. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

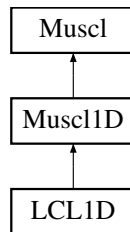
Parameters

in	<i>n</i>	Label of the eigenvalue
out	<i>v</i>	Vect instance containing the corresponding eigenvector. This vector is resized.

7.56 LCL1D Class Reference

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

Inheritance diagram for LCL1D:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `LCL1D (Mesh &m)`
Constructor using mesh instance.
- `LCL1D (Mesh &m, Vect< real.t > &U)`
Constructor.
- `~LCL1D ()`
Destructor.
- `Vect< real.t > & getFlux ()`
Return sidewise fluxes.
- `void setInitialCondition (Vect< real.t > &u)`
Assign initial condition by a vector.
- `void setInitialCondition (real.t u)`
Assign a constant initial condition.
- `void setReconstruction ()`
Run MUSCL reconstruction.
- `real.t runOneTimeStep ()`
Run one time step of the linear conservation law.
- `void setBC (real.t u)`
Set Dirichlet boundary condition.
- `void setBC (const Side &sd, real.t u)`
Set Dirichlet boundary condition.
- `void setBC (int code, real.t u)`
Set Dirichlet boundary condition.
- `void setVelocity (Vect< real.t > &v)`
Set convection velocity.
- `void setVelocity (real.t v)`
Set (constant) convection velocity.
- `void setReferenceLength (real.t dx)`
Assign reference length value.
- `real.t getReferenceLength () const`
Return reference length.
- `void Forward (const Vect< real.t > &Flux, Vect< real.t > &Field)`
Computation of the primal variable $n \rightarrow n+1$.
- `real.t getMeanLength () const`
Return mean length.
- `real.t getMaximumLength () const`
Return maximal length.
- `real.t getMinimumLength () const`
Return mimal length.
- `real.t getTauLim () const`
Return mean length.
- `void print_mesh_stat ()`
Output mesh information.
- `void setTimeStep (real.t dt)`

- Assign time step value.*

 - `real_t getTimeStep ()` const

Return time step value.
- `void setCFL (real_t CFL)`

Assign CFL value.
- `real_t getCFL ()` const

Return CFL value.
- `Mesh & getMesh ()` const

Return reference to `Mesh` instance.
- `void setVerbose (int v)`

Set verbosity parameter.
- `bool setReconstruction (const Vect< real_t > &U, Vect< real_t > &LU, Vect< real_t > &RU, size_t dof)`

Function to reconstruct by the `Muscl` method.
- `void setMethod (const Method &s)`

Choose a flux solver.
- `void setSolidZoneCode (int c)`

Choose a code for solid zone.
- `bool getSolidZone ()` const

Return flag for presence of solid zones.
- `int getSolidZoneCode ()` const

Return code of solid zone, 0 if this one is not present.
- `void setLimiter (Limiter l)`

Choose a flux limiter.

7.56.1 Detailed Description

Class to solve the linear conservation law (Hyperbolic equation) in 1-D by a MUSCL Finite Volume scheme.

7.56.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method

MULTI_SLOPE_Q_METHOD Multislope Q method

MULTI_SLOPE_M_METHOD Multislope M method

M_FORCE_WORD for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver
VFROE_SOLVER Finite Volume Roe solver
LF_SOLVER LF solver
RUSANOV_SOLVER Rusanov solver
HLL_SOLVER HLL solver
HLLC_SOLVER HLLC solver
MAX_SOLVER Max solver
FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.56.3 Member Function Documentation**void setInitialCondition (Vect< real_t > & u)**

Assign initial condition by a vector.

Parameters

in	<i>u</i>	Vector containing initial condition
-----------	----------	-------------------------------------

void setInitialCondition (real_t u)

Assign a constant initial condition.

Parameters

in	<i>u</i>	Constant value for the initial condition
-----------	----------	--

real_t runOneTimeStep ()

Run one time step of the linear conservation law.

Returns

Value of the time step

void setBC (real_t u)

Set Dirichlet boundary condition.

Assign a constant value *u* to all boundary sides

void setBC (const Side & sd, real_t u)

Set Dirichlet boundary condition.

Assign a constant value to a side

Parameters

in	<i>sd</i>	Side to which value is prescribed
in	<i>u</i>	Value to prescribe

void setBC (int code, real_t u)

Set Dirichlet boundary condition.

Assign a constant value sides with a given code

Parameters

in	<i>code</i>	Code of sides to which value is prescribed
in	<i>u</i>	Value to prescribe

void setVelocity (Vect< real_t > & v)

Set convection velocity.

Parameters

in	<i>v</i>	Vect instance containing velocity
----	----------	---

void Forward (const Vect< real_t > & Flux, Vect< real_t > & Field)

Computation of the primal variable $n \rightarrow n+1$.

Vector **Flux** contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, **getNeighborElement(1)** and right element **getNeighborElement(2)** if **getNeighborElement(2)** doesn't exist, we are on a boundary and we prescribe a symmetry condition

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	v	Value of verbosity parameter
----	---	------------------------------

bool setReconstruction (const Vect< real.t > & U, Vect< real.t > & LU, Vect< real.t > & RU, size_t dof) [inherited]

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	s	Solver to choose
----	---	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

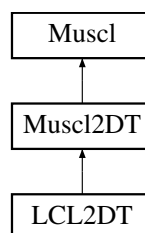
Parameters

in	/	Limiter to choose
----	---	-------------------

7.57 LCL2DT Class Reference

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

Inheritance diagram for LCL2DT:



Public Types

- enum [Method](#) {
[FIRST_ORDER_METHOD](#) = 0,
[MULTI_SLOPE_Q_METHOD](#) = 1,
[MULTI_SLOPE_M_METHOD](#) = 2,
[M_FORCE_WORD](#) = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `LCL2DT (Mesh &m)`
Constructor using `Mesh` instance.
- `LCL2DT (Mesh &m, Vect< real_t > &U)`
Constructor using mesh and initial data.
- `~LCL2DT ()`
Destructor.
- `Vect< real_t > & getFlux ()`
Return sidewise flux vector.
- `void setInitialCondition (Vect< real_t > &u)`
Set elementwise initial condition.
- `void setInitialCondition (real_t u)`
Set a constant initial condition.
- `void setReconstruction ()`
Reconstruct flux using `Muscl` scheme.
- `real_t runOneTimeStep ()`
Run one time step of the linear conservation law.
- `void setBC (real_t u)`
Set Dirichlet boundary condition.
- `void setBC (const Side &sd, real_t u)`
Set Dirichlet boundary condition.
- `void setBC (int code, real_t u)`
Set Dirichlet boundary condition.
- `void setVelocity (const Vect< real_t > &v)`
Set convection velocity.
- `void setVelocity (const LocalVect< real_t, 2 > &v)`
Set (constant) convection velocity.
- `void Forward (const Vect< real_t > &Flux, Vect< real_t > &Field)`
Computation of the primal variable $n \rightarrow n+1$.

- bool `setReconstruction` (const `Vect< real_t >` &U, `Vect< real_t >` &LU, `Vect< real_t >` &RU, size_t dof)
Function to reconstruct by the `Muscl` method.
- void `setTimeStep` (`real_t` dt)
Assign time step value.
- `real_t` `getTimeStep` () const
Return time step value.
- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getCFL` () const
Return CFL value.
- void `setReferenceLength` (`real_t` dx)
Assign reference length value.
- `real_t` `getReferenceLength` () const
Return reference length.
- `Mesh` & `getMesh` () const
Return reference to `Mesh` instance.
- void `setVerbose` (int v)
Set verbosity parameter.
- void `setMethod` (const `Method` &s)
Choose a flux solver.
- void `setSolidZoneCode` (int c)
Choose a code for solid zone.
- bool `getSolidZone` () const
Return flag for presence of solid zones.
- int `getSolidZoneCode` () const
Return code of solid zone, 0 if this one is not present.
- void `setLimiter` (`Limiter` l)
Choose a flux limiter.

Protected Member Functions

- void `Initialize` ()
Construction of normals to sides.

7.57.1 Detailed Description

Class to solve the linear hyperbolic equation in 2-D by a MUSCL Finite Volume scheme on triangles.

7.57.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method

MULTI_SLOPE_Q_METHOD Multislope Q method

MULTI_SLOPE_M_METHOD Multislope M method

M_FORCE_WORD for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter

VANLEER_LIMITER Van Leer limiter

SUPERBEE_LIMITER Superbee limiter

VANALBADA_LIMITER Van Albada limiter

MAX_LIMITER Max limiter

L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver

VFROE_SOLVER Finite Volume Roe solver

LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.57.3 Constructor & Destructor Documentation

LCL2DT (**Mesh** & **m**, **Vect**< **real_t** > & **U**)

Constructor using mesh and initial data.

Parameters

in	m	Reference to Mesh instance
in	U	Vector containing initial (elementwise) solution

7.57.4 Member Function Documentation

void setInitialCondition (**Vect**< **real_t** > & **u**)

Set elementwise initial condition.

Parameters

in	u	Vect instance containing initial condition values
-----------	----------	---

void setInitialCondition (**real_t** **u**)

Set a constant initial condition.

Parameters

in	<i>u</i>	Value of initial condition to assign to all elements
-----------	----------	--

real.t runOneTimeStep ()

Run one time step of the linear conservation law.

Returns

Value of the time step

void setBC (real.t u)

Set Dirichlet boundary condition.

Assign a constant value *u* to all boundary sides

void setBC (const Side & sd, real.t u)

Set Dirichlet boundary condition.

Assign a constant value to a side

Parameters

in	<i>sd</i>	Side to which value is prescribed
in	<i>u</i>	Value to prescribe

void setBC (int code, real.t u)

Set Dirichlet boundary condition.

Assign a constant value sides with a given code

Parameters

in	<i>code</i>	Code of sides to which value is prescribed
in	<i>u</i>	Value to prescribe

void setVelocity (const Vect< real.t > & v)

Set convection velocity.

Parameters

in	<i>v</i>	Vect instance containing velocity
-----------	----------	---

void setVelocity (const LocalVect< real.t, 2 > & v)

Set (constant) convection velocity.

Parameters

in	<i>v</i>	Vector containing constant velocity to prescribe
-----------	----------	--

void Forward (const Vect< real.t > & Flux, Vect< real.t > & Field)

Computation of the primal variable $n \rightarrow n+1$.

Vector *Flux* contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, [getNeighborElement\(1\)](#) and right element [getNeighborElement\(2\)](#) if [getNeighborElement\(2\)](#) doesn't exist, we are on a boundary and we prescribe a symmetry condition

```
bool setReconstruction ( const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t >  
& RU, size_t dof ) [inherited]
```

Function to reconstruct by the [Muscl](#) method.

Parameters

in	U	Field to reconstruct
out	LU	Left gradient vector
out	RU	Right gradient vector
in	dof	Label of dof to reconstruct

void Initialize () [protected], [inherited]

Construction of normals to sides.

Convention: for a given side, `getPtrElement(1)` is the left element and `getPtrElement(2)` is the right element. The normal goes from left to right. For boundary sides, the normal points outward.

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	dt	Time step value
----	------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	CFL	Value of CFL
----	-------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	dx	Value of reference length
----	------	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	v	Value of verbosity parameter
----	-----	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	s	Solver to choose
----	-----	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

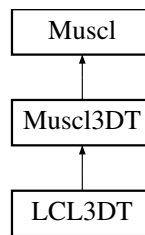
Parameters

in	/	Limiter to choose
----	---	-------------------

7.58 LCL3DT Class Reference

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

Inheritance diagram for LCL3DT:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `LCL3DT` (`Mesh` &m)
Constructor using mesh.
- `LCL3DT` (`Mesh` &m, `Vect`< `real.t` > &U)

- Constructor using mesh and initial field.*
 - `~LCL3DT ()`
 - Destructor.*
- `void setInitialCondition (Vect< real_t > &u)`
 - Set elementwise initial condition.*
- `void setInitialCondition (real_t u)`
 - Set a constant initial condition.*
- `void setReconstruction ()`
 - Reconstruct flux using *Muscl* scheme.*
- `real_t runOneTimeStep ()`
 - Run one time step.*
- `void setBC (real_t u)`
 - Set Dirichlet boundary condition. Assign a constant value *u* to all boundary sides.*
- `void setBC (const Side &sd, real_t u)`
 - Set Dirichlet boundary condition.*
- `void setBC (int code, real_t u)`
 - Set Dirichlet boundary condition.*
- `void setVelocity (const Vect< real_t > &v)`
 - Set convection velocity.*
- `void setVelocity (const LocalVect< real_t, 3 > &v)`
 - Set (constant) convection velocity.*
- `void setReferenceLength (real_t dx)`
 - Assign reference length value.*
- `real_t getReferenceLength () const`
 - Return reference length.*
- `void Forward (const Vect< real_t > &Flux, Vect< real_t > &Field)`
 - Computation of the primal variable $n \rightarrow n+1$.*
- `bool setReconstruction (const Vect< real_t > &U, Vect< real_t > &LU, Vect< real_t > &RU, size_t dof)`
 - Function to reconstruct by the *Muscl* method.*
- `real_t getMinimumFaceArea () const`
 - Return minimum area of faces in the mesh.*
- `real_t getMinimumElementVolume () const`
 - Return minimum volume of elements in the mesh.*
- `real_t getMaximumFaceArea () const`
 - Return maximum area of faces in the mesh.*
- `real_t getMaximumElementVolume () const`
 - Return maximum volume of elements in the mesh.*
- `real_t getMeanFaceArea () const`
 - Return mean area of faces in the mesh.*
- `real_t getMeanElementVolume () const`
 - Return mean volume of elements in the mesh.*
- `real_t getMinimumEdgeLength () const`
 - Return minimum length of edges in the mesh.*
- `real_t getMinimumVolumebyArea () const`
 - Return minimum volume by area in the mesh.*
- `real_t getMaximumEdgeLength () const`

- Return maximum length of edges in the mesh.*
 - `real_t getTauLim ()` const
Return value of tau lim.
 - `real_t getComega ()` const
Return value of Comega.
 - `void setbetalim (real_t bl)`
Assign value of beta lim.
 - `void setTimeStep (real_t dt)`
Assign time step value.
 - `real_t getTimeStep ()` const
Return time step value.
 - `void setCFL (real_t CFL)`
Assign CFL value.
 - `real_t getCFL ()` const
Return CFL value.
 - `Mesh & getMesh ()` const
Return reference to [Mesh](#) instance.
 - `void setVerbose (int v)`
Set verbosity parameter.
 - `void setMethod (const Method &s)`
Choose a flux solver.
 - `void setSolidZoneCode (int c)`
Choose a code for solid zone.
 - `bool getSolidZone ()` const
Return flag for presence of solid zones.
 - `int getSolidZoneCode ()` const
Return code of solid zone, 0 if this one is not present.
 - `void setLimiter (Limiter l)`
Choose a flux limiter.

7.58.1 Detailed Description

Class to solve the linear conservation law equation in 3-D by a MUSCL Finite Volume scheme on tetrahedra.

7.58.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

- FIRST_ORDER_METHOD** First Order upwind method
- MULTI_SLOPE_Q_METHOD** Multislope Q method
- MULTI_SLOPE_M_METHOD** Multislope M method
- M_FORCE_WORD** for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter

VANLEER_LIMITER Van Leer limiter

SUPERBEE_LIMITER Superbee limiter

VANALBADA_LIMITER Van Albada limiter

MAX_LIMITER Max limiter

L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver

VFROE_SOLVER Finite Volume Roe solver

LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.58.3 Constructor & Destructor Documentation

LCL3DT (**Mesh** & *m*, **Vect**< **real_t** > & *U*)

Constructor using mesh and initial field.

Parameters

<i>in</i>	<i>m</i>	Reference to Mesh instance
<i>in</i>	<i>U</i>	Vector containing initial (elementwise) solution

7.58.4 Member Function Documentation

void setInitialCondition (**Vect**< **real_t** > & *u*)

Set elementwise initial condition.

Parameters

<i>in</i>	<i>u</i>	Vect instance containing initial condition values
-----------	----------	---

void setInitialCondition (**real_t** *u*)

Set a constant initial condition.

Parameters

<code>in</code>	<code>u</code>	Value of initial condition to assign to all elements
-----------------	----------------	--

void setBC (const Side & sd, real_t u)

Set Dirichlet boundary condition.

Assign a constant value to a side

Parameters

<code>in</code>	<code>sd</code>	Side to which value is prescribed
<code>in</code>	<code>u</code>	Value to prescribe

void setBC (int code, real_t u)

Set Dirichlet boundary condition.

Assign a constant value sides with a given code

Parameters

<code>in</code>	<code>code</code>	Code of sides to which value is prescribed
<code>in</code>	<code>u</code>	Value to prescribe

void setVelocity (const Vect< real_t > & v)

Set convection velocity.

Parameters

<code>in</code>	<code>v</code>	Vect instance containing velocity
-----------------	----------------	---

void setVelocity (const LocalVect< real_t, 3 > & v)

Set (constant) convection velocity.

Parameters

<code>in</code>	<code>v</code>	Vector containing constant velocity to prescribe
-----------------	----------------	--

void Forward (const Vect< real_t > & Flux, Vect< real_t > & Field)

Computation of the primal variable $n \rightarrow n+1$.

Vector Flux contains elementwise fluxes issued from the Riemann problem, calculated with, as left element, **getNeighborElement(1)** and right element **getNeighborElement(2)** if **getNeighborElement(2)** doesn't exist, we are on a boundary and we prescribe a symmetry condition

bool setReconstruction (const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t > & RU, size_t dof) [inherited]

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void setTimeStep (real.t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real.t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	<i>v</i>	Value of verbosity parameter
----	----------	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	<i>s</i>	Solver to choose
----	----------	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

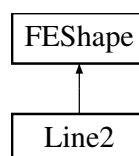
Parameters

in	<i>l</i>	Limiter to choose
----	----------	-------------------

7.59 Line2 Class Reference

To describe a 2-Node planar line finite element.

Inheritance diagram for Line2:



Public Member Functions

- `Line2 ()`
Default Constructor.
- `Line2 (const Element *el)`
Constructor for an element.
- `Line2 (const Side *side)`
Constructor for a side.
- `Line2 (const Edge *edge)`
Constructor for an edge.
- `~Line2 ()`
Destructor.
- `real_t getLength () const`
Return element length.
- `Point< real_t > getNormal () const`
Return unit normal vector to line.
- `Point< real_t > getTangent () const`
Return unit tangent vector to line.
- `real_t Sh (size_t i, real_t s) const`
Calculate shape function of a given node at a given point.
- `real_t DSh (size_t i) const`
Calculate partial derivative of shape function of a node.
- `Point< real_t > getRefCoord (const Point< real_t > &x)`
Return reference coordinates of a point x in element.
- `bool isIn (const Point< real_t > &x)`
Check whether point x is in current line element or not.
- `real_t getInterpolate (const Point< real_t > &x, const LocalVect< real_t, 2 > &v)`
Return interpolated value at a given point.
- `real_t Sh (size_t i) const`
Return shape function of node i at given point.
- `real_t Sh (size_t i, Point< real_t > s) const`
Calculate shape function of node i at a given point s .
- `real_t getDet () const`
Return determinant of jacobian.
- `Point< real_t > getCenter () const`
Return coordinates of center of element.
- `Point< real_t > getLocalPoint () const`
Localize a point in the element.
- `Point< real_t > getLocalPoint (const Point< real_t > &s) const`
Localize a point in the element.

7.59.1 Detailed Description

To describe a 2-Node planar line finite element.

Defines geometric quantities associated to 2-node linear segment element P_1 in the space. The reference element is the segment $[-1, 1]$. Note that the line length is not checked unless the function check is called.

7.59.2 Constructor & Destructor Documentation

Line2 (const Element * el)

Constructor for an element.

Parameters

<i>in</i>	<i>el</i>	Pointer to element
-----------	-----------	--------------------

Line2 (const Side * side)

Constructor for a side.

Parameters

<i>in</i>	<i>side</i>	Pointer to side
-----------	-------------	-----------------

Line2 (const Edge * edge)

Constructor for an edge.

Parameters

<i>in</i>	<i>edge</i>	Pointer to edge
-----------	-------------	-----------------

7.59.3 Member Function Documentation**real_t Sh (size_t i, real_t s) const**

Calculate shape function of a given node at a given point.

Parameters

<i>in</i>	<i>i</i>	Node number (1 or 2).
<i>in</i>	<i>s</i>	Localization of point in natural coordinates (must be between -1 and 1).

real_t DSh (size_t i) const

Calculate partial derivative of shape function of a node.

Parameters

<i>in</i>	<i>i</i>	Node number (1 or 2).
-----------	----------	---------------------------------------

Point<real_t> getRefCoord (const Point< real_t > & x)

Return reference coordinates of a point x in element.

Only the x-coordinate of the returned value has a meaning

real_t getInterpolate (const Point< real_t > & x, const LocalVect< real_t, 2 > & v)

Return interpolated value at a given point.

Parameters

<i>in</i>	<i>x</i>	Point where interpolation is evaluated (in the reference element).
<i>out</i>	<i>v</i>	Computed value.

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

in	<i>i</i>	Local node label
in	<i>s</i>	Point in the reference triangle where the shape function is evaluated

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

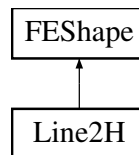
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.60 Line2H Class Reference

To describe a 2-Node Hermite planar line finite element.

Inheritance diagram for Line2H:



Public Member Functions

- [Line2H](#) ()
Default Constructor.
- [Line2H](#) (const [Element](#) *el)
Constructor for an element.
- [Line2H](#) (const [Side](#) *side)
Constructor for a side.
- [~Line2H](#) ()
Destructor.
- [Point< real_t > getLocalPoint](#) ([real_t](#) s) const
Localize a point in the element.
- [real_t Sh](#) (size_t i, [real_t](#) s) const
Return shape function value of node i at given point s
- [real_t DSh](#) (size_t i, [real_t](#) s) const
Return first derivative (along the abscissa) of shape function of node i at a given point.
- [real_t D2Sh](#) (size_t i, [real_t](#) s) const

- Return second derivatives (along the abscissa) of shape function of node i*

 - `real_t getDet ()` const
Return determinant of jacobian.
 - `real_t getLength ()`
Return element length.
 - `real_t check ()` const
Check line length and number of line nodes.
 - `real_t Sh (size_t i)` const
Return shape function of node i at given point.
 - `real_t Sh (size_t i, Point< real_t > s)` const
Calculate shape function of node i at a given point s .
 - `Point< real_t > DSh (size_t i)` const
Return derivatives of shape function of node i at a given point.
 - `Point< real_t > getCenter ()` const
Return coordinates of center of element.
 - `Point< real_t > getLocalPoint ()` const
Localize a point in the element.
 - `Point< real_t > getLocalPoint (const Point< real_t > &s)` const
Localize a point in the element.

7.60.1 Detailed Description

To describe a 2-Node Hermite planar line finite element.

Defines geometric quantities associated to 2-node segment element in the space using Hermite (C^1) interpolation. The interpolation functions are polynomials of degree 3. The reference element is the segment $[-1, 1]$. The unknowns are supported by extremities of the interval: each extremity supports two unknowns, the function and its line derivative.

7.60.2 Member Function Documentation

Point<real_t> getLocalPoint (real_t s) const

Localize a point in the element.

For a point s in the reference element, return coordinates in the real element.

real_t check () const

Check line length and number of line nodes.

Returns

- > 0 : m is the length
- $= 0$: zero length (\Rightarrow Error)

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s .

Parameters

in	i	Local node label
in	s	Point in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const [inherited]

Return derivatives of shape function of node i at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

Parameters

in	i	Partial derivative index (1, 2 or 3)
----	---	--------------------------------------

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

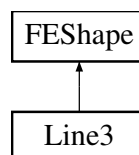
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.61 Line3 Class Reference

To describe a 3-Node quadratic planar line finite element.

Inheritance diagram for Line3:



Public Member Functions

- [Line3](#) ()
Default Constructor.
- [Line3](#) (const [Element](#) *el)
Constructor for an element.
- [Line3](#) (const [Side](#) *sd)
Constructor for a side.
- [~Line3](#) ()
Destructor.
- void [setLocal](#) (double s)
Initialize local point coordinates in element.
- double [DSh](#) (size_t i) const
Return derivatives of shape function of node i at a given point.

- `Point< double > getLocalPoint ()` const
Return actual coordinates of localized point.
- `real_t Sh (size_t i)` const
Return shape function of node i at given point.
- `real_t Sh (size_t i, Point< real_t > s)` const
Calculate shape function of node i at a given point s .
- `real_t getDet ()` const
Return determinant of jacobian.
- `Point< real_t > getCenter ()` const
Return coordinates of center of element.
- `Point< real_t > getLocalPoint (const Point< real_t > &s)` const
Localize a point in the element.

7.61.1 Detailed Description

To describe a 3-Node quadratic planar line finite element.

Defines geometric quantities associated to 3-node quadratic element P_2 in the space. The reference element is the segment $[-1, 1]$. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function `setLocal()` must be invoked.

`Element` nodes are ordered as the following: the left one, the central one and the right one.

7.61.2 Member Function Documentation

`real_t Sh (size_t i, Point< real_t > s)` const [inherited]

Calculate shape function of node i at a given point s .

Parameters

in	i	Local node label
in	s	<code>Point</code> in the reference triangle where the shape function is evaluated

`real_t getDet ()` const [inherited]

Return determinant of jacobian.

If the transformation (Reference element \rightarrow Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

`Point<real_t> getLocalPoint (const Point< real_t > &s)` const [inherited]

Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.62 LocalMatrix< T_, NR_, NC_ > Class Template Reference

Handles small size matrices like element matrices, with a priori known size.

Public Member Functions

- [LocalMatrix](#) ()
Default constructor.
- [LocalMatrix](#) (const [LocalMatrix](#)< T_, NR_, NC_ > &m)
Copy constructor.
- [LocalMatrix](#) (class [Element](#) *el, const [SpMatrix](#)< T_ > &a)
Constructor of a local matrix associated to element from a [SpMatrix](#).
- [LocalMatrix](#) (class [Element](#) *el, const [SkMatrix](#)< T_ > &a)
Constructor of a local matrix associated to element from a [SkMatrix](#).
- [LocalMatrix](#) (class [Element](#) *el, const [SkSMatrix](#)< T_ > &a)
Constructor of a local matrix associated to element from a [SkSMatrix](#).
- [~LocalMatrix](#) ()
Destructor.
- T_ & [operator\(\)](#) (size_t i, size_t j)
Operator () (Non constant version)
- T_ [operator\(\)](#) (size_t i, size_t j) const
Operator () (Constant version)
- void [Localize](#) (class [Element](#) *el, const [SpMatrix](#)< T_ > &a)
Initialize matrix as element matrix from global [SpMatrix](#).
- void [Localize](#) (class [Element](#) *el, const [SkMatrix](#)< T_ > &a)
Initialize matrix as element matrix from global [SkMatrix](#).
- void [Localize](#) (class [Element](#) *el, const [SkSMatrix](#)< T_ > &a)
Initialize matrix as element matrix from global [SkSMatrix](#).
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator=](#) (const [LocalMatrix](#)< T_, NR_, NC_ > &m)
Operator =
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator=](#) (const T_ &x)
Operator =
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator+=](#) (const [LocalMatrix](#)< T_, NR_, NC_ > &m)
Operator +=
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator-=](#) (const [LocalMatrix](#)< T_, NR_, NC_ > &m)
Operator -=
- [LocalVect](#)< T_, NR_ > [operator*](#) ([LocalVect](#)< T_, NC_ > &x)
*Operator **
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator+=](#) (const T_ &x)
Operator +=
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator-=](#) (const T_ &x)
Operator -=
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator*=](#) (const T_ &x)
*Operator *=*
- [LocalMatrix](#)< T_, NR_, NC_ > & [operator/=](#) (const T_ &x)
Operator /=
- void [MultAdd](#) (const [LocalVect](#)< T_, NC_ > &x, [LocalVect](#)< T_, NR_ > &y)
Multiply matrix by vector and add result to vector.
- void [MultAddScal](#) (const T_ &a, const [LocalVect](#)< T_, NC_ > &x, [LocalVect](#)< T_, NR_ > &y)
Multiply matrix by scaled vector and add result to vector.
- void [Mult](#) (const [LocalVect](#)< T_, NC_ > &x, [LocalVect](#)< T_, NR_ > &y)

- `void Symmetrize ()`
Symmetrize matrix.
- `int Factor ()`
Factorize matrix.
- `int Solve (LocalVect< T_, NR_ > &b)`
Forward and backsubstitute to solve a linear system.
- `int FactorAndSolve (LocalVect< T_, NR_ > &b)`
Factorize matrix and solve linear system.
- `void Invert (LocalMatrix< T_, NR_, NC_ > &A)`
Calculate inverse of matrix.
- `T_ getInnerProduct (const LocalVect< T_, NC_ > &x, const LocalVect< T_, NR_ > &y)`
Calculate inner product with respect to matrix.
- `T_ * get ()`
Return pointer to matrix as a C-array.

7.62.1 Detailed Description

template<class T_, size_t NR_, size_t NC_>class OFELI::LocalMatrix< T_, NR_, NC_ >

Handles small size matrices like element matrices, with a priori known size.

The template class `LocalMatrix` treats small size matrices. Typically, this class is recommended to store element and side arrays.

Internally, no dynamic storage is used.

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
<code><NR_></code>	number of rows of matrix
<code><NC_></code>	number of columns of matrix

7.62.2 Constructor & Destructor Documentation

LocalMatrix ()

Default constructor.

Constructs a matrix with 0 rows and 0 columns

LocalMatrix (class Element * el, const SpMatrix< T_ > & a)

Constructor of a local matrix associated to element from a `SpMatrix`.

Parameters

<code>in</code>	<code>el</code>	Pointer to <code>Element</code>
<code>in</code>	<code>a</code>	Global matrix as instance of class <code>SpMatrix</code> .

LocalMatrix (class Element * el, const SkMatrix< T_ > & a)

Constructor of a local matrix associated to element from a `SkMatrix`.

Parameters

<code>in</code>	<code>el</code>	Pointer to Element
<code>in</code>	<code>a</code>	Global matrix as instance of class SkMatrix .

LocalMatrix (class Element * el, const SkSMatrix< T_ > & a)

Constructor of a local matrix associated to element from a [SkSMatrix](#).

Parameters

<code>in</code>	<code>el</code>	Pointer to Element
<code>in</code>	<code>a</code>	Global matrix as instance of class SkSMatrix .

7.62.3 Member Function Documentation

T_& operator() (size_t i, size_t j)

Operator () (Non constant version)

Returns entry at row i and column j.

T_ operator() (size_t i, size_t j) const

Operator () (Constant version)

Returns entry at row i and column j.

void Localize (class Element * el, const SpMatrix< T_ > & a)

Initialize matrix as element matrix from global [SpMatrix](#).

Parameters

<code>in</code>	<code>el</code>	Pointer to Element
<code>in</code>	<code>a</code>	Global matrix as instance of class SpMatrix . This function is called by its corresponding constructor.

void Localize (class Element * el, const SkMatrix< T_ > & a)

Initialize matrix as element matrix from global [SkMatrix](#).

Parameters

<code>in</code>	<code>el</code>	Pointer to Element
<code>in</code>	<code>a</code>	Global matrix as instance of class SkMatrix . This function is called by its corresponding constructor.

void Localize (class Element * el, const SkSMatrix< T_ > & a)

Initialize matrix as element matrix from global [SkSMatrix](#).

Parameters

<code>in</code>	<code>el</code>	Pointer to Element
<code>in</code>	<code>a</code>	Global matrix as instance of class SkSMatrix . This function is called by its corresponding constructor.

LocalMatrix< T_, NR_, NC_ > & operator= (const LocalMatrix< T_, NR_, NC_ > & m)

Operator =

Copy instance *m* into current instance.

LocalMatrix< T_, NR_, NC_ > & operator= (const T_ & x)

Operator =

Assign *x* to diagonal matrix entries. The resulting matrix is diagonal.

LocalMatrix< T_, NR_, NC_ > & operator+= (const LocalMatrix< T_, NR_, NC_ > & m)

Operator +=

Add *m* to current matrix.

LocalMatrix< T_, NR_, NC_ > & operator-= (const LocalMatrix< T_, NR_, NC_ > & m)

Operator -=

Subtract *m* from current matrix.

LocalVect< T_, NR_ > operator* (LocalVect< T_, NC_ > & x)

Operator *

Return a [Vect](#) instance as product of current matrix by vector *x*.

LocalMatrix< T_, NR_, NC_ > & operator+= (const T_ & x)

Operator +=

Add constant *x* to current matrix entries.

LocalMatrix< T_, NR_, NC_ > & operator-= (const T_ & x)

Operator -=

Subtract *x* from current matrix entries.

LocalMatrix< T_, NR_, NC_ > & operator*= (const T_ & x)

Operator *=

Multiply matrix entries by constant *x*.

LocalMatrix< T_, NR_, NC_ > & operator/= (const T_ & x)

Operator /=

Divide by *x* current matrix entries.

void MultAdd (const LocalVect< T_, NC_ > & x, LocalVect< T_, NR_ > & y)

Multiply matrix by vector and add result to vector.

Parameters

in	<i>x</i>	Vector to multiply matrix by.
out	<i>y</i>	Resulting vector (<i>y</i> += <i>a</i> * <i>x</i>)

void MultAddScal (const T_ & a, const LocalVect< T_, NC_ > & x, LocalVect< T_, NR_ > & y)

Multiply matrix by scaled vector and add result to vector.

Parameters

in	a	Constant to premultiply by vector x .
in	x	(Scaled) vector to multiply matrix by.
out	y	Resulting vector ($y += a * x$)

void Mult (const LocalVect< T_, NC_ > & x, LocalVect< T_, NR_ > & y)

Multiply matrix by vector.

Parameters

in	x	Vector to multiply matrix by.
out	y	Resulting vector.

void Symmetrize ()

Symmetrize matrix.

Fill upper triangle to form a symmetric matrix.

int Factor ()

Factorize matrix.

Performs a LU factorization.

Returns

- 0: Factorization has ended normally,
- n : n -th pivot was zero.

int Solve (LocalVect< T_, NR_ > & b)

Forward and backsubstitute to solve a linear system.

Parameters

in	b	Right-hand side in input and solution vector in output.
----	-----	---

Returns

- 0: Solution was performed normally.
- n : n -th pivot is zero.

Note

Matrix must have been factorized at first.

int FactorAndSolve (LocalVect< T_, NR_ > & b)

Factorize matrix and solve linear system.

Parameters

in,out	<i>b</i>	Right-hand side in input and solution vector in output.
--------	----------	---

Returns

0 if solution was performed normally. *n* if *n*-th pivot is zero. This function simply calls [Factor\(\)](#) then **Solve(b)**.

void Invert (LocalMatrix< T_, NR_, NC_ > & A)

Calculate inverse of matrix.

Parameters

out	<i>A</i>	Inverse of matrix
-----	----------	-------------------

T_ getInnerProduct (const LocalVect< T_, NC_ > & x, const LocalVect< T_, NR_ > & y)

Calculate inner product with respect to matrix.

Returns the product $x^T Ay$

Parameters

in	<i>x</i>	Left vector
in	<i>y</i>	Right vector

Returns

Resulting product

7.63 LocalVect< T_, N_ > Class Template Reference

Handles small size vectors like element vectors.

Public Member Functions

- [LocalVect](#) ()
Default constructor.
- [LocalVect](#) (const T_ *a)
Constructor using a C-array.
- [LocalVect](#) (const class [Element](#) *el)
Constructor using [Element](#) pointer.
- [LocalVect](#) (const class [Side](#) *sd)
Constructor using [Side](#) pointer.
- [LocalVect](#) (const [LocalVect](#)< T_, N_ > &v)
Copy constructor.
- [LocalVect](#) (const class [Element](#) *el, const class [Vect](#)< T_ > &v, int opt=0)
Constructor of an element vector from a global [Vect](#) instance.
- [LocalVect](#) (const class [Side](#) *sd, const class [Vect](#)< T_ > &v, int opt=0)
Constructor of a side vector from a global [Vect](#) instance.
- [~LocalVect](#) ()
Destructor.
- void [getLocal](#) (const class [Element](#) &el, const class [Vect](#)< T_ > &v, int type)

- Localize an element vector from a global Vect instance.*
 - void **Localize** (const class **Element** *el, const class **Vect**< T_ > &v, size_t k=0)
- Localize an element vector from a global Vect instance.*
 - void **Localize** (const class **Side** *sd, const class **Vect**< T_ > &v, size_t k=0)
- Localize a side vector from a global Vect instance.*
 - T_ & **operator[]** (size_t i)
- Operator [] (Non constant version).*
 - T_ **operator[]** (size_t i) const
- Operator [] (Constant version).*
 - T_ & **operator()** (size_t i)
- Operator () (Non constant version).*
 - T_ **operator()** (size_t i) const
- Operator () (Constant version).*
 - **Element** * **El** ()
- Return pointer to Element if vector was constructed using an element and NULL otherwise.*
 - **Side** * **Sd** ()
- Return pointer to Side if vector was constructed using a side and NULL otherwise.*
 - **LocalVect**< T_, N_ > & **operator=** (const **LocalVect**< T_, N_ > &v)
- Operator =*
 - **LocalVect**< T_, N_ > & **operator=** (const T_ &x)
- Operator =*
 - **LocalVect**< T_, N_ > & **operator+=** (const **LocalVect**< T_, N_ > &v)
- Operator +=*
 - **LocalVect**< T_, N_ > & **operator+=** (const T_ &a)
- Operator +=*
 - **LocalVect**< T_, N_ > & **operator-=** (const **LocalVect**< T_, N_ > &v)
- Operator -=*
 - **LocalVect**< T_, N_ > & **operator-=** (const T_ &a)
- Operator -=*
 - **LocalVect**< T_, N_ > & **operator*=** (const T_ &a)
- Operator *=*
 - **LocalVect**< T_, N_ > & **operator/=** (const T_ &a)
- Operator /=*
 - T_ * **get** ()
- Return pointer to vector as a C-Array.*
 - T_ **operator**, (const **LocalVect**< T_, N_ > &v) const
- Return Dot (scalar) product of two vectors.*

7.63.1 Detailed Description

template<class T_, size_t N_>class OFELI::LocalVect< T_, N_ >

Handles small size vectors like element vectors.

The template class **LocalVect** treats small size vectors. Typically, this class is recommended to store element and side arrays. Operators =, [] and () are overloaded so that one can write for instance:

```
LocalVect<double,10> u, v;
v = -1.0;
u = v;
u(3) = -2.0;
```

to set vector **v** entries to -1, copy vector **v** into vector **u** and assign third entry of **v** to -2. Notice that entries of **v** are here **v(1)**, **v(2)**, ..., **v(10)**, *i.e.* vector entries start at index 1. Internally, no dynamic storage is used.

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
<code><N_></code>	Vector size

7.63.2 Constructor & Destructor Documentation

LocalVect (const class Element * el, const class Vect< T_ > & v, int opt = 0)

Constructor of an element vector from a global Vect instance.

The constructed vector has local numbering of nodes

Parameters

in	<i>el</i>	Pointer to Element to localize
in	<i>v</i>	Global vector to localize
in	<i>opt</i>	Option for DOF treatment <ul style="list-style-type: none"> • = 0, Normal case [Default] • Any other value : only one DOF is handled (Local vector has as dimension number of degrees of freedom)

LocalVect (const class Side * sd, const class Vect< T_ > & v, int opt = 0)

Constructor of a side vector from a global Vect instance.

The constructed vector has local numbering of nodes

Parameters

in	<i>sd</i>	Pointer to Side to localize
in	<i>v</i>	Global vector to localize
in	<i>opt</i>	Option for DOF treatment <ul style="list-style-type: none"> • = 0, Normal case [Default] • Any other value : only one DOF is handled (Local vector has as dimension number of degrees of freedom)

7.63.3 Member Function Documentation

void getLocal (const class Element & el, const class Vect< T_ > & v, int type)

Localize an element vector from a global Vect instance.

The constructed vector has local numbering of nodes This function is called by the constructor:

LocalVect(const Element *el, const Vect<T_> &v)

Parameters

in	<i>el</i>	Pointer to Element to localize
in	<i>v</i>	Global vector to localize
in	<i>type</i>	Type of element. This is to be chosen among enumerated values: LINE2, TRIANG3, QUAD4, TETRA4, HEXA8, PENTA6

void Localize (const class Element * el, const class Vect< T_ > & v, size_t k = 0)

Localize an element vector from a global Vect instance.

The constructed vector has local numbering of nodes This function is called by the constructor:
LocalVect(const Element *el, const Vect<T_> &v)

Parameters

in	<i>el</i>	Pointer to Side to localize
in	<i>v</i>	Global vector to localize
in	<i>k</i>	Degree of freedom to localize [Default: All degrees of freedom are stored]

void Localize (const class Side *sd, const class Vect< T_ > &v, size_t k = 0)

Localize a side vector from a global [Vect](#) instance.

The constructed vector has local numbering of nodes This function is called by the constructor:
LocalVect(const Side *sd, const Vect<T_> &v)

Parameters

in	<i>sd</i>	Pointer to Side to localize
in	<i>v</i>	Global vector to localize
in	<i>k</i>	Degree of freedom to localize [Default: All degrees of freedom are stored]

T_& operator[] (size_t i)

Operator [] (Non constant version).

v[i] starts at *v[0]* to *v[size()-1]*

T_ operator[] (size_t i) const

Operator [] (Constant version).

v[i] starts at *v[0]* to *v[size()-1]*

T_& operator() (size_t i)

Operator () (Non constant version).

v(i) starts at *v(1)* to *v(size())*. *v(i)* is the same element as *v[i-1]*

T_ operator() (size_t i) const

Operator () (Constant version).

v(i) starts at *v(1)* to *v(size())* *v(i)* is the same element as *v[i-1]*

LocalVect< T_, N_ > & operator= (const LocalVect< T_, N_ > &v)

Operator =

Copy a [LocalVect](#) instance to the current one

LocalVect< T_, N_ > & operator= (const T_ &x)

Operator =

Assign value *x* to all vector entries

LocalVect< T_, N_ > & operator+= (const LocalVect< T_, N_ > &v)

Operator +=

Add vector *v* to this instance

LocalVect< T_, N_ > & operator+= (const T_ & a)

Operator +=

Add constant a to vector entries

LocalVect< T_, N_ > & operator-= (const LocalVect< T_, N_ > & v)

Operator -=

Subtract vector v from this instance

LocalVect< T_, N_ > & operator-= (const T_ & a)

Operator -=

Subtract constant a from vector entries

LocalVect< T_, N_ > & operator*= (const T_ & a)

Operator *=

Multiply vector by constant a

LocalVect< T_, N_ > & operator/= (const T_ & a)

Operator /=

Divide vector by constant a

T_ operator, (const LocalVect< T_, N_ > & v) const

Return Dot (scalar) product of two vectors.

A typical use of this operator is double a = (v,w) where v and w are 2 instances of LocalVect<double,n>

Parameters

in	v	LocalVect instance by which the current instance is multiplied
----	---	--

7.64 Material Class Reference

To treat material data. This class enables reading material data in material data files. It also returns these informations by means of its members.

Public Member Functions

- [Material](#) ()
Default constructor.
- [Material](#) (const [Material](#) &m)
Copy constructor.
- [~Material](#) ()
Destructor.
- int [set](#) (int m, const string &name)
Associate to material code number n the material named name
- string [getName](#) (int m) const
Return material name for material with code m
- int [getCode](#) (size_t i) const
Return material code for i-th material.

- `size_t getNbMat ()` `const`
Return Number of read materials.
- `void setCode (int m)`
Associate code m to current material.
- `int check (int c)`
- `real_t Density ()`
Return constant density.
- `real_t Density (const Point< real_t > &x, real_t t)`
Return density at point x and time t
- `real_t SpecificHeat ()`
Return constant specific heat.
- `real_t SpecificHeat (const Point< real_t > &x, real_t t)`
Return specific heat at point x and time t
- `real_t ThermalConductivity ()`
Return constant thermal conductivity.
- `real_t ThermalConductivity (const Point< real_t > &x, real_t t)`
Return thermal conductivity at point x and time t
- `real_t MeltingTemperature ()`
Return constant melting temperature.
- `real_t MeltingTemperature (const Point< real_t > &x, real_t t)`
Return melting temperature at point x and time t
- `real_t EvaporationTemperature ()`
Return constant evaporation temperature.
- `real_t EvaporationTemperature (const Point< real_t > &x, real_t t)`
Return evaporation temperature at point x and time t
- `real_t ThermalExpansion ()`
Return constant thermal expansion coefficient.
- `real_t ThermalExpansion (const Point< real_t > &x, real_t t)`
Return thermal expansion coefficient at point x and time t
- `real_t LatentHeatForMelting ()`
Return constant latent heat for melting.
- `real_t LatentHeatForMelting (const Point< real_t > &x, real_t t)`
Return latent heat for melting at point x and time t
- `real_t LatentHeatForEvaporation ()`
Return constant latent heat for evaporation.
- `real_t LatentHeatForEvaporation (const Point< real_t > &x, real_t t)`
Return latent heat for evaporation at point x and time t
- `real_t DielectricConstant ()`
Return constant dielectric constant.
- `real_t DielectricConstant (const Point< real_t > &x, real_t t)`
Return dielectric constant at point x and time t
- `real_t ElectricConductivity ()`
Return constant electric conductivity.
- `real_t ElectricConductivity (const Point< real_t > &x, real_t t)`
Return electric conductivity at point x and time t
- `real_t ElectricResistivity ()`

- Return constant electric resistivity.*
 - `real_t ElectricResistivity` (const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return electric resistivity at point `x` and time `t`*
 - `real_t MagneticPermeability` ()
 - Return constant magnetic permeability.*
 - `real_t MagneticPermeability` (const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return magnetic permeability at point `x` and time `t`*
 - `real_t Viscosity` ()
 - Return constant viscosity.*
 - `real_t Viscosity` (const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return viscosity at point `x` and time `t`*
 - `real_t YoungModulus` ()
 - Return constant Young modulus.*
 - `real_t YoungModulus` (const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return Young modulus at point `x` and time `t`*
 - `real_t PoissonRatio` ()
 - Return constant Poisson ratio.*
 - `real_t PoissonRatio` (const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return Poisson ratio at point `x` and time `t`*
 - `real_t Property` (int `i`)
 - Return constant *i*-th property.*
 - `real_t Property` (int `i`, const `Point`< `real_t` > &`x`, `real_t` `t`)
 - Return *i*-th property at point `x` and time `t`*
 - `Material` & `operator=` (const `Material` &`m`)
 - Operator =.*

7.64.1 Detailed Description

To treat material data. This class enables reading material data in material data files. It also returns these informations by means of its members.

7.64.2 Constructor & Destructor Documentation

`Material` ()

Default constructor.

It initializes the class and searches for the path where are material data files.

7.64.3 Member Function Documentation

`int set` (int `m`, const string & `name`)

Associate to material code number `n` the material named `name`

Returns

Number of materials

`string getName` (int `m`) const

Return material name for material with code `m`

If such a material is not found, return a blank string.

int check (int c)

Check if material code c is present.

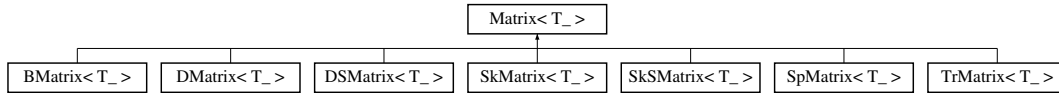
Returns

0 if succeeded, 1 if not.

7.65 Matrix< T_ > Class Template Reference

Virtual class to handle matrices for all storage formats.

Inheritance diagram for Matrix< T_ >:



Public Member Functions

- **Matrix ()**
Default constructor.
- **Matrix (const Matrix< T_ > &m)**
Copy Constructor.
- **virtual ~Matrix ()**
Destructor.
- **size_t getNbRows () const**
Return number of rows.
- **size_t getNbColumns () const**
Return number of columns.
- **void setPenal (real_t p)**
Set Penalty Parameter (For boundary condition prescription).
- **void setDiagonal ()**
Set the matrix as diagonal.
- **T_ getDiag (size_t k) const**
Return k-th diagonal entry of matrix.
- **size_t size () const**
Return matrix dimension (Number of rows and columns).
- **virtual void MultAdd (const Vect< T_ > &x, Vect< T_ > &y) const =0**
Multiply matrix by vector x and add to y
- **virtual void MultAdd (T_ a, const Vect< T_ > &x, Vect< T_ > &y) const =0**
*Multiply matrix by vector a*x and add to y*
- **virtual void Mult (const Vect< T_ > &x, Vect< T_ > &y) const =0**
Multiply matrix by vector x and save in y
- **virtual void TMult (const Vect< T_ > &v, Vect< T_ > &w) const =0**
Multiply transpose of matrix by vector x and save in y
- **virtual void Apxy (T_ a, const Matrix< T_ > *x)=0**
Add to matrix the product of a matrix by a scalar.
- **void setDiagonal (const class Mesh &mesh)**

- Initialize matrix storage in the case where only diagonal terms are stored.*
- void [Assembly](#) (const class [Element](#) *el, T_ *a)
Assembly of element matrix into global matrix.
- void [Assembly](#) (const class [Element](#) *el, const [DMatrix](#)< T_ > &a)
Assembly of element matrix into global matrix.
- void [Assembly](#) (const class [Side](#) *sd, T_ *a)
Assembly of side matrix into global matrix.
- void [Assembly](#) (const class [Side](#) *sd, const [DMatrix](#)< T_ > &a)
Assembly of side matrix into global matrix.
- void [Prescribe](#) (const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition.
- void [Prescribe](#) (int dof, int code, const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- void [Prescribe](#) ([Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.
- void [Prescribe](#) (const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition.
- void [Prescribe](#) ([Vect](#)< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.
- void [Prescribe](#) (size_t dof, const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void [Prescribe](#) (size_t dof, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void [PrescribeSide](#) (const class [Mesh](#) &mesh)
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void [PrescribeSide](#) ()
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- virtual void [add](#) (size_t i, size_t j, const T_ &val)=0
Add val to entry (i, j).
- virtual int [Factor](#) ()=0
Factorize matrix. Available only if the storage class enables it.
- virtual int [Solve](#) ([Vect](#)< T_ > &b)=0
Solve the linear system by a direct method.
- int [Solve](#) (const [Vect](#)< T_ > &b, [Vect](#)< T_ > &x)
Solve system with factorized matrix (forward and back substitution).
- int [FactorAndSolve](#) ([Vect](#)< T_ > &b)
Factorize matrix and solve the linear system.
- int [FactorAndSolve](#) (const [Vect](#)< T_ > &b, [Vect](#)< T_ > &x)
Factorize matrix and solve the linear system.
- unsigned long [getLength](#) () const
Return number of stored terms in matrix.
- int [isDiagonal](#) () const

- Say if matrix is diagonal or not.*

 - int `isFactorized` () const
- Say if matrix is factorized or not.*

 - virtual size_t `getCollInd` (size_t i) const

Return Column index for column i (See the description for class `SpMatrix`).
- virtual size_t `getRowPtr` (size_t i) const

Return Row pointer for row i (See the description for class `SpMatrix`).
- virtual void `set` (size_t i, size_t j, const T_ &val)=0

Assign a value to an entry of the matrix.
- virtual T_ & `operator()` (size_t i, size_t j)=0

Operator () (Non constant version).
- virtual T_ `operator()` (size_t i, size_t j) const =0

Operator () (Non constant version).
- T_ `operator()` (size_t i) const

Operator () with one argument (Constant version).
- T_ & `operator()` (size_t i)

Operator () with one argument (Non Constant version).
- T_ & `operator[]` (size_t k)

Operator [] (Non constant version).
- T_ `operator[]` (size_t k) const

Operator [] (Constant version).
- `Matrix` & `operator=` (`Matrix`< T_ > &m)

Operator =.
- `Matrix` & `operator+=` (const `Matrix`< T_ > &m)

Operator +=.
- `Matrix` & `operator-=` (const `Matrix`< T_ > &m)

Operator -=.
- `Matrix` & `operator=` (const T_ &x)

Operator =.
- `Matrix` & `operator*=` (const T_ &x)

*Operator *=.*
- `Matrix` & `operator+=` (const T_ &x)

Operator +=.
- `Matrix` & `operator-=` (const T_ &x)

Operator -=.
- virtual T_ `get` (size_t i, size_t j) const =0

Return entry (i, j) of matrix if this one is stored, 0 else.
- void `setPrintView` (size_t rmin, size_t rmax, size_t cmin, size_t cmax)

Set a window for matrix printing.
- void `getPrintView` (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const

Return window data for matrix printing.

7.65.1 Detailed Description

`template<class T_>class OFELI::Matrix< T_ >`

Virtual class to handle matrices for all storage formats.

Template Parameters

<code><T_></code>	Data type (real_t, float, complex<real_t>, ...)
-------------------------	---

7.65.2 Constructor & Destructor Documentation

Matrix ()

Default constructor.

Initializes a zero-size matrix.

7.65.3 Member Function Documentation

T_ getDiag (size_t k) const

Return k-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

virtual void Axy (T_ a, const Matrix< T_ > * x) [pure virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>x</i>	Matrix by which <i>a</i> is multiplied. The result is added to current instance

Implemented in [SpMatrix< T_ >](#), [SkSMatrix< T_ >](#), [DMatrix< T_ >](#), [DSMatrix< T_ >](#), [SkMatrix< T_ >](#), [TrMatrix< T_ >](#), and [BMatrix< T_ >](#).

void setDiagonal (const class Mesh & mesh)

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

void Assembly (const class Element * el, T_ * a)

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a)

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a)

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a)

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.

<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).
-----------	-------------	--

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0)

Impose by a penalty method a homogeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0)

Impose by a penalty method a homogeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<code>in</code>	<code>dof</code>	Label of the concerned degree of freedom (DOF).
<code>in</code>	<code>mesh</code>	Mesh instance from which information is extracted.
<code>in,out</code>	<code>b</code>	Vect instance that contains right-hand side.
<code>in</code>	<code>u</code>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<code>in</code>	<code>flag</code>	Parameter to determine whether only the right-hand side is to be modified (<code>dof>0</code>) or both matrix and right-hand side (<code>dof=0</code> , default value).

void Prescribe (`size_t` dof, [Vect](#)< T_ > & b, const [Vect](#)< T_ > & u, int flag = 0)

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<code>in</code>	<code>dof</code>	Label of the concerned degree of freedom (DOF).
<code>in,out</code>	<code>b</code>	Vect instance that contains right-hand side.
<code>in</code>	<code>u</code>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<code>in</code>	<code>flag</code>	Parameter to determine whether only the right-hand side is to be modified (<code>dof>0</code>) or both matrix and right-hand side (<code>dof=0</code> , default value).

void PrescribeSide (const class [Mesh](#) & mesh)

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide ()

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

virtual int Solve ([Vect](#)< T_ > & b) [pure virtual]

Solve the linear system by a direct method.

This is available only if the storage class enables it and if matrix has been primarily factorized (See `isFactorized`).

Implemented in [SpMatrix< T_ >](#), [SkMatrix< T_ >](#), [SkSMatrix< T_ >](#), [DMatrix< T_ >](#), [DSMatrix< T_ >](#), [TrMatrix< T_ >](#), and [BMatrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve system with factorized matrix (forward and back substitution).

Parameters

in	b	Vect instance that contains right-hand side
out	x	Vect instance that contains solution

Returns

- - 0 if solution was normally performed
 - n if the n-th pivot is null
- Solution is performed only if factorization has previously been invoked.

int FactorAndSolve (Vect< T_ > & b)

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	b	Vect instance that contains right-hand side on input and solution on output
--------	---	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x)

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	b	Vect instance that contains right-hand side
out	x	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- n if the n-th pivot is null

int isFactorized () const

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

virtual void set (size_t i, size_t j, const T_ & val) [pure virtual]

Assign a value to an entry of the matrix.

Parameters

<code>in</code>	<code>i</code>	Row index
<code>in</code>	<code>j</code>	Column index
<code>in</code>	<code>val</code>	Value to assign

Implemented in [SpMatrix< T_ >](#), [SkSMatrix< T_ >](#), [SkMatrix< T_ >](#), [TrMatrix< T_ >](#), [DMatrix< T_ >](#), [BMatrix< T_ >](#), and [DSMatrix< T_ >](#).

virtual T_& operator() (size_t i, size_t j) [pure virtual]

Operator () (Non constant version).

Returns the (i,j) entry of the matrix.

Parameters

<code>in</code>	<code>i</code>	Row index
<code>in</code>	<code>j</code>	Column index

Implemented in [SpMatrix< T_ >](#), [SkSMatrix< T_ >](#), [SkMatrix< T_ >](#), [DMatrix< T_ >](#), [TrMatrix< T_ >](#), [BMatrix< T_ >](#), and [DSMatrix< T_ >](#).

virtual T_ operator() (size_t i, size_t j) const [pure virtual]

Operator () (Non constant version).

Returns the (i,j) entry of the matrix.

Parameters

<code>in</code>	<code>i</code>	Row index
<code>in</code>	<code>j</code>	Column index

Implemented in [SpMatrix< T_ >](#), [SkSMatrix< T_ >](#), [SkMatrix< T_ >](#), [DMatrix< T_ >](#), [TrMatrix< T_ >](#), [BMatrix< T_ >](#), and [DSMatrix< T_ >](#).

T_ operator() (size_t i) const

Operator () with one argument (Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

<code>in</code>	<code>i</code>	entry index
-----------------	----------------	-------------

T_ & operator() (size_t i)

Operator () with one argument (Non Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

<code>in</code>	<code>i</code>	entry index
-----------------	----------------	-------------

T_ & operator[] (size_t k)

Operator [] (Non constant version).

Returns k-th stored element in matrix Index k starts at 0.

T_ operator[] (size_t k) const

Operator [] (Constant version).

Returns k-th stored element in matrix Index k starts at 0.

Matrix< T_ > & operator= (Matrix< T_ > & m)

Operator =.

Copy matrix m to current matrix instance.

Matrix< T_ > & operator+= (const Matrix< T_ > & m)

Operator +=.

Add matrix m to current matrix instance.

Matrix< T_ > & operator-= (const Matrix< T_ > & m)

Operator -=.

Subtract matrix m from current matrix instance.

Matrix< T_ > & operator= (const T_ & x)

Operator =.

Assign constant value x to all matrix entries.

Matrix< T_ > & operator*= (const T_ & x)

Operator *.

Premultiply matrix entries by constant value x

Matrix< T_ > & operator+= (const T_ & x)

Operator +=.

Add constant value x to all matrix entries.

Matrix< T_ > & operator-= (const T_ & x)

Operator -=.

Subtract constant value x from all matrix entries.

void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const

Return window data for matrix printing.

Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.66 Mesh Class Reference

To store and manipulate finite element meshes.

Public Member Functions

- [Mesh](#) ()
Default constructor (Empty mesh)
- [Mesh](#) (const string &file, bool bc=false, int opt=NODE_DOF)
Constructor using a mesh file.
- [Mesh](#) (real_t L, size_t nb_el, size_t p=1, size_t nb_dof=1)
Constructor for a 1-D mesh. The domain is the interval [0,L].
- [Mesh](#) (const [Grid](#) &g, int opt=QUADRILATERAL)
Constructor for a uniform finite difference grid given by and instance of class [Grid](#).
- [Mesh](#) (const [Grid](#) &g, int shape, int opt)
Constructor of dual mesh for a uniform finite difference grid given by and instance of class [Grid](#).
- [Mesh](#) (real_t xL, real_t yL, size_t nx, size_t ny, int c1, int c2, int c3, int c4, int opt=0)
Constructor for a uniform finite difference grid.
- [Mesh](#) (const [Mesh](#) &m, const [Point](#)< real_t > &x_bl, const [Point](#)< real_t > &x_tr)
Constructor that extracts the mesh of a rectangular region from an initial mesh.
- [Mesh](#) (const [Mesh](#) &mesh, int opt, size_t dof1, size_t dof2, bool bc=false)
Constructor that copies the input mesh and selects given degrees of freedom.
- [Mesh](#) (const [Mesh](#) &ms)
Copy Constructor.
- [~Mesh](#) ()
Destructor.
- void [setDim](#) (size_t dim)
Define space dimension. Normally, between 1 and 3.
- void [setVerbose](#) (int verb)
Define Verbose Parameter. Controls output details.
- void [Add](#) ([Node](#) *nd)
Add a node to mesh.
- void [Add](#) ([Element](#) *el)
Add an element to mesh.
- void [Add](#) ([Side](#) *sd)
Add a side to mesh.
- void [Add](#) ([Edge](#) *ed)
Add an edge to mesh.
- [Mesh](#) & [operator*=](#) (real_t a)
*Operator *=*

- void `get` (const string &mesh_file)
Read mesh data in file.
- void `get` (const string &mesh_file, int ff, int nb_dof=1)
Read mesh data in file with giving its format.
- void `setDOFSupport` (int opt, int nb_nodes=1)
Define supports of degrees of freedom.
- void `setNbDOFPerNode` (size_t nb_dof=1)
Define number of degrees of freedom for each node.
- void `setPointInDomain` (Point< real_t > x)
Define a point in the domain. This function makes sense only if boundary mesh is given without internal mesh (Case of Boundary Elements)
- void `removeImposedDOF` ()
Eliminate equations corresponding to imposed DOF.
- size_t `NumberEquations` (size_t dof=0)
Renumber Equations.
- size_t `NumberEquations` (size_t dof, int c)
Renumber Equations.
- int `getAllSides` (int opt=0)
Determine all mesh sides.
- int `getNbSideNodes` () const
Return the number of nodes on each side.
- int `getNbElementNodes` () const
Return the number of nodes in each element.
- int `getBoundarySides` ()
Determine all boundary sides.
- int `createBoundarySideList` ()
Create list of boundary sides.
- int `getBoundaryNodes` ()
Determine all boundary nodes.
- int `createInternalSideList` ()
Create list of internal sides (not on the boundary).
- int `getAllEdges` ()
Determine all edges.
- void `getNodeNeighborElements` ()
Create node neighboring elements.
- void `getElementNeighborElements` ()
Create element neighboring elements.
- void `setMaterial` (int code, const string &mname)
Associate material to code of element.
- void `Reorder` (size_t m=GRAPH_MEMORY)
Renumber mesh nodes according to reverse Cuthill Mc Kee algorithm.
- void `Add` (size_t num, real_t *x)
Add a node by giving its label and an array containing its coordinates.
- void `DeleteNode` (size_t label)
Remove a node given by its label.
- void `DeleteElement` (size_t label)
Remove an element given by its label.

- void `DeleteSide` (size_t label)
Remove a side given by its label.
- void `Delete` (Node *nd)
Remove a node given by its pointer.
- void `Delete` (Element *el)
Remove a node given by its pointer.
- void `Delete` (Side *sd)
Remove a side given by its pointer.
- void `Delete` (Edge *ed)
Remove an edge given by its pointer.
- void `RenumberNode` (size_t n1, size_t n2)
Renumber a node.
- void `RenumberElement` (size_t n1, size_t n2)
Renumber an element.
- void `RenumberSide` (size_t n1, size_t n2)
Renumber a side.
- void `RenumberEdge` (size_t n1, size_t n2)
Renumber an edge.
- void `setNodeView` (size_t n1, size_t n2)
Set viewing window for nodes.
- void `setElementView` (size_t n1, size_t n2)
Set viewing window for elements.
- void `setSideView` (size_t n1, size_t n2)
Set viewing window for sides.
- void `setEdgeView` (size_t n1, size_t n2)
Set viewing window for edges.
- void `setList` (const std::vector< Node * > &n1)
Initialize list of mesh nodes using the input vector.
- void `setList` (const std::vector< Element * > &el)
Initialize list of mesh elements using the input vector.
- void `setList` (const std::vector< Side * > &sl)
Initialize list of mesh sides using the input vector.
- void `Rescale` (real_t sx, real_t sy=0., real_t sz=0.)
Rescale mesh by multiplying node coordinates by constants.
- int `getVerbose` () const
Return Verbose Parameter.
- size_t `getDim` () const
Return space dimension.
- size_t `getNbNodes` () const
Return number of nodes.
- size_t `getNbMarkedNodes` () const
Return number of marked nodes.
- size_t `getNbVertices` () const
Return number of vertices.
- size_t `getNbDOF` () const
Return total number of degrees of freedom (DOF)

- size_t `getNbEq ()` const
Return number of equations.
- size_t `getNbEq (int i)` const
Return number of equations for the i-th set of degrees of freedom.
- size_t `getNbElements ()` const
Return number of elements.
- size_t `getNbSides ()` const
Return number of sides.
- size_t `getNbEdges ()` const
Return number of sides.
- size_t `getNbBoundarySides ()` const
Return number of boundary sides.
- size_t `getNbInternalSides ()` const
Return number of internal sides.
- size_t `getNbMat ()` const
Return number of materials.
- void `AddMidNodes (int g=0)`
Add mid-side nodes.
- `Point< real_t > getMaxCoord ()` const
Return maximum coordinates of nodes.
- `Point< real_t > getMinCoord ()` const
Return minimum coordinates of nodes.
- void `set (Node *nd)`
Replace node in the mesh.
- void `set (Element *el)`
Replace element in the mesh.
- void `set (Side *sd)`
Choose side in the mesh.
- bool `NodesAreDOF ()` const
Return information about DOF type.
- bool `SidesAreDOF ()` const
Return information about DOF type.
- bool `EdgesAreDOF ()` const
Return information about DOF type.
- bool `ElementsAreDOF ()` const
Return information about DOF type.
- int `getDOFSupport ()` const
Return information on dof support Return an integer according to enumerated values: NODE_DOF, ELEMENT_DOF SIDE_DOF.
- void `put (const string &mesh_file)` const
Write mesh data on file.
- void `save (const string &mesh_file)` const
Write mesh data on file in various formats.
- void `Bput (const string &mesh_file)` const
Write mesh data on a binary file.
- bool `withImposedDOF ()` const
Return true if imposed DOF count in equations, false if not.

- `bool isStructured ()` const
Return true is mesh is structured, false if not.
- `size_t getNodeNewLabel (size_t n)` const
Return new label of node of a renumbered node.
- `void getList (vector< Node * > &nl)` const
Fill vector nl with list of pointers to nodes.
- `void getList (vector< Element * > &el)` const
Fill vector el with list of pointers to elements.
- `void getList (vector< Side * > &sl)` const
Fill vector sl with list of pointers to sides.
- `Node * getPtrNode (size_t i)` const
Return pointer to node with label i.
- `Node & getNode (size_t i)` const
Return referrece to node with label i
- `Element * getPtrElement (size_t i)` const
Return pointer to element with label i
- `Element & getElement (size_t i)` const
Return reference to element with label i
- `Side * getPtrSide (size_t i)` const
Return pointer to side with label i
- `Side & getSide (size_t i)` const
Return reference to side with label i
- `Edge * getPtrEdge (size_t i)` const
Return pointer to edge with label i
- `Edge & getEdge (size_t i)` const
Return reference to edge with label i
- `size_t getNodeLabel (size_t i)` const
Return label of i-th node.
- `size_t getElementLabel (size_t i)` const
Return label of i-th element.
- `size_t getSideLabel (size_t i)` const
Return label of i-th side.
- `size_t getEdgeLabel (size_t i)` const
Return label of i-th edge.
- `void topNode ()` const
Reset list of nodes at its top position (Non constant version)
- `void topBoundaryNode ()` const
Reset list of boundary nodes at its top position (Non constant version)
- `void topMarkedNode ()` const
Reset list of marked nodes at its top position (Non constant version)
- `void topElement ()` const
Reset list of elements at its top position (Non constant version)
- `void topSide ()` const
Reset list of sides at its top position (Non constant version)
- `void topBoundarySide ()` const
Reset list of boundary sides at its top position (Non constant version)

- void `topInternalSide ()` const
Reset list of internal sides at its top position (Non constant version)
- void `topEdge ()` const
Reset list of edges at its top position (Non constant version)
- void `topBoundaryEdge ()` const
Reset list of boundary edges at its top position (Non constant version)
- `Node *` `getNode ()` const
Return pointer to current node and move to next one (Non constant version)
- `Node *` `getBoundaryNode ()` const
Return pointer to current boundary node and move to next one (Non constant version)
- `Node *` `getMarkedNode ()` const
Return pointer to current marked node and move to next one (Non constant version)
- `Element *` `getElement ()` const
Return pointer to current element and move to next one (Non constant version)
- `Element *` `getActiveElement ()` const
Return pointer to current element and move to next one (Non constant version)
- `Side *` `getSide ()` const
Return pointer to current side and move to next one (Non constant version)
- `Side *` `getBoundarySide ()` const
Return pointer to current boundary side and move to next one (Non constant version)
- `Side *` `getInternalSide ()` const
Return pointer to current internal side and move to next one (Non constant version)
- `Edge *` `getEdge ()` const
Return pointer to current edge and move to next one (Non constant version)
- `Edge *` `getBoundaryEdge ()` const
Return pointer to current boundary edge and move to next one (Non constant version)
- int `getShape ()` const
Determine shape of elements Return Shape index (see enum ElementShape) if all elements have the same shape, 0 if not.
- `Element *` `operator() (size_t i)` const
Operator () : Return pointer to i-th element.
- `Node *` `operator[] (size_t i)` const
Operator [] : Return pointer to i-th node.
- size_t `operator() (size_t i, size_t n)` const
Operator () : Return pointer to i-th node of n-th element.
- `Mesh &` `operator= (Mesh &ms)`
Operator = : Assign a Mesh instance.

Friends

- void `Refine (Mesh &in_mesh, Mesh &out_mesh)`
Refine mesh. Subdivide each triangle into 4 subtriangles. This member function is valid for 2-D triangular meshes only.

7.66.1 Detailed Description

To store and manipulate finite element meshes.

Class [Mesh](#) enables defining as an object a finite element mesh. A finite element mesh is characterized by its nodes, elements and sides. Each of these types of data constitutes a class in the [OFELI](#) library.

The standard procedure to introduce the finite element mesh is to provide an input file containing its data. For this, we have defined our own mesh data file (following the XML syntax). Of course, a developer can write his own function to read his finite element mesh file using the methods in [Mesh](#).

7.66.2 Constructor & Destructor Documentation

Mesh (const string & file, bool bc = false, int opt = NODE_DOF)

Constructor using a mesh file.

Parameters

in	file	File containing mesh data
in	bc	Flag to remove (true) or not (false) imposed Degrees of Freedom [default: false]
in	opt	Type of DOF support: To choose among enumerated values NODE_DOF, SIDE_DOF or ELEMENT_DOF. Say if degrees of freedom (unknowns) are supported by nodes, sides or elements.

Mesh (real_t L, size_t nb_el, size_t p = 1, size_t nb_dof = 1)

Constructor for a 1-D mesh. The domain is the interval [0,L].

Parameters

in	L	Length of the interval
in	nb_el	Number of elements to generate
in	p	Degree of finite element polynomial (Default = 1)
in	nb_dof	Number of degrees of freedom for each node (Default = 1)

Mesh (const Grid & g, int opt = QUADRILATERAL)

Constructor for a uniform finite difference grid given by and instance of class [Grid](#).

Parameters

in	g	Grid instance
in	opt	Optional value to say which type of elements to generate <ul style="list-style-type: none"> • TRIANGLE: Mesh elements are triangles • QUADRILATERAL: Mesh elements are quadrilaterals [default]

Mesh (const Grid & g, int shape, int opt)

Constructor of dual mesh for a uniform finite difference grid given by and instance of class [Grid](#).

Parameters

<code>in</code>	<code>g</code>	Grid instance
<code>in</code>	<code>shape</code>	Value to say which type of elements to generate <ul style="list-style-type: none"> • TRIANGLE: Mesh elements are triangles • QUADRILATERAL: Mesh elements are quadrilaterals [default]
<code>in</code>	<code>opt</code>	This argument can take any value. It is here only to distinguish from the other constructor using Grid instance.

Remarks

This constructor is to be used to obtain a dual mesh from a structured grid. It is mainly useful if a cell centered finite volume method is used.

Mesh (`real_t xL`, `real_t yL`, `size_t nx`, `size_t ny`, `int c1`, `int c2`, `int c3`, `int c4`, `int opt = 0`)

Constructor for a uniform finite difference grid.

The domain is the rectangle $(0,L_x) \times (0,L_y)$

Parameters

<code>in</code>	<code>xL</code>	Length in the x-direction
<code>in</code>	<code>yL</code>	Length in the y-direction
<code>in</code>	<code>nx</code>	Number of subintervals on the x-axis
<code>in</code>	<code>ny</code>	Number of subintervals on the y-axis
<code>in</code>	<code>c1</code>	Code for nodes generated on the line $y=0$
<code>in</code>	<code>c2</code>	Code for nodes generated on the line $x=L_x$
<code>in</code>	<code>c3</code>	Code for nodes generated on the line $y=L_y$
<code>in</code>	<code>c4</code>	Code for nodes generated on the line $x=0$
<code>in</code>	<code>opt</code>	Flag to generate elements as well (if not zero) [Default: 0]. If the flag is not 0, it can take one of the enumerated values: TRIANGLE or QUADRILATERAL, with obvious meaning.

Mesh (`const Mesh & m`, `const Point< real_t > & x_bl`, `const Point< real_t > & x_tr`)

Constructor that extracts the mesh of a rectangular region from an initial mesh.

This constructor is useful for zooming purposes for instance.

Parameters

<code>in</code>	<code>m</code>	Initial mesh from which the submesh is extracted
<code>in</code>	<code>x_bl</code>	Coordinate of bottom left vertex of the rectangle
<code>in</code>	<code>x_tr</code>	Coordinate of top right vertex of the rectangle

Mesh (`const Mesh & mesh`, `int opt`, `size_t dof1`, `size_t dof2`, `bool bc = false`)

Constructor that copies the input mesh and selects given degrees of freedom.

This constructor is to be used for coupled problems where each subproblem uses a choice of degrees of freedom.

Parameters

in	<i>mesh</i>	Initial mesh from which the submesh is extracted
in	<i>opt</i>	Type of DOF support: To choose among enumerated values <code>NODE_DOF</code> , <code>SIDE_DOF</code> or <code>ELEMENT_DOF</code> .
in	<i>dof1</i>	Label of first degree of freedom to select to the output mesh
in	<i>dof2</i>	Label of last degree of freedom to select to the output mesh
in	<i>bc</i>	Flag to remove (true) or not (false) imposed Degrees of Freedom [Default: false]

Mesh (const Mesh & ms)

Copy Constructor.

Parameters

in	<i>ms</i>	Mesh instance to copy
----	-----------	---------------------------------------

7.66.3 Member Function Documentation**void setDim (size_t dim)**

Define space dimension. Normally, between 1 and 3.

Parameters

in	<i>dim</i>	Space dimension to set (must be between 1 and 3)
----	------------	--

void setVerbose (int verb)

Define Verbose Parameter. Controls output details.

Parameters

in	<i>verb</i>	verbosity parameter (Must be between 0 and 10)
----	-------------	--

void Add (Node * nd)

Add a node to mesh.

Parameters

in	<i>nd</i>	Pointer to Node to add
----	-----------	--

void Add (Element * el)

Add an element to mesh.

Parameters

in	<i>el</i>	Pointer to Element to add
----	-----------	---

void Add (Side * sd)

Add a side to mesh.

Parameters

<i>in</i>	<i>sd</i>	Pointer to Side to add
-----------	-----------	--

void Add (Edge * ed)

Add an edge to mesh.

Parameters

<i>in</i>	<i>ed</i>	Pointer to Edge to add
-----------	-----------	--

Mesh& operator*= (real_t a)

Operator *=

Rescale mesh coordinates by multiplying by a factor

Parameters

<i>in</i>	<i>a</i>	Value to multiply by
-----------	----------	----------------------

void get (const string & mesh_file)

Read mesh data in file.

[Mesh](#) file must be in [OFELI](#) format. See "File Formats" page

Parameters

<i>in</i>	<i>mesh_file</i>	Mesh file name
-----------	------------------	--------------------------------

void get (const string & mesh_file, int ff, int nb_dof = 1)

Read mesh data in file with giving its format.

File format can be chosen among a variety of choices. See "File Formats" page

Parameters

<i>in</i>	<i>mesh_file</i>	Mesh file name
<i>in</i>	<i>ff</i>	File format: Integer to chose among enumerated values: OFELI_FF , GMSH , MATLAB , EASYMESH , GAMBIT , BAMG , NETGEN , TRIANGLE_FF
<i>in</i>	<i>nb_dof</i>	Number of degrees of freedom per node (Default value: 1)

void setDOFSupport (int opt, int nb_nodes = 1)

Define supports of degrees of freedom.

Parameters

<i>in</i>	<i>opt</i>	DOF type: <ul style="list-style-type: none"> • NODE_DOF: Degrees of freedom are supported by nodes • SIDE_DOF: Degrees of freedom are supported by sides • EDGE_DOF: Degrees of freedom are supported by edges • ELEMENT_DOF: Degrees of freedom are supported by elements
<i>in</i>	<i>nb_nodes</i>	Number of nodes on sides or elements (default=1). This parameter is useful only if dofs are supported by sides or elements

Note

This member function creates all mesh sides if the option `ELEMENT_DOF` or `SIDE_DOF` is selected. So it not necessary to call `getAllSides()` after

void setNbDOFPerNode (size_t nb_dof = 1)

Define number of degrees of freedom for each node.

Parameters

<code>in</code>	<code>nb_dof</code>	Number of degrees of freedom (unknowns) for each mesh node (Default value is 1)
-----------------	---------------------	---

Note

This function first declares nodes as unknown supports, sets the number of degrees of freedom and rennumbers equations

void setPointInDomain (Point< real.t > x)

Define a point in the domain. This function makes sense only if boundary mesh is given without internal mesh (Case of Boundary Elements)

Parameters

<code>in</code>	<code>x</code>	Coordinates of point to define
-----------------	----------------	--------------------------------

size_t NumberEquations (size_t dof = 0)

Renumber Equations.

Parameters

<code>in</code>	<code>dof</code>	Label of degree of freedom for which numbering is performed. Default value (0) means that all degrees of freedom are taken into account
-----------------	------------------	---

size_t NumberEquations (size_t dof, int c)

Renumber Equations.

Parameters

<code>in</code>	<code>dof</code>	Label of degree of freedom for which numbering is performed.
<code>in</code>	<code>c</code>	code for which degrees of freedom are enforced.

int getAllSides (int opt = 0)

Determine all mesh sides.

Returns

Number of all sides.

int getBoundarySides ()

Determine all boundary sides.

Returns

Number of boundary sides.

int createBoundarySideList ()

Create list of boundary sides.

This function is useful to loop over boundary sides without testing. Once this one is called, the function [getNbBoundarySides\(\)](#) is available. Moreover, looping over boundary sides is available via the member functions [topBoundarySide\(\)](#) and [getBoundarySide\(\)](#)

Returns

Number of boundary sides.

int getBoundaryNodes ()

Determine all boundary nodes.

Returns

n Number of boundary nodes.

int createInternalSideList ()

Create list of internal sides (not on the boundary).

This function is useful to loop over internal sides without testing. Once this one is called, the function [getNbInternalSides\(\)](#) is available. Moreover, looping over internal sides is available via the member functions [topInternalSide\(\)](#) and [getInternalSide\(\)](#)

Returns

n Number of internal sides.

int getAllEdges ()

Determine all edges.

Returns

Number of all edges.

void getNodeNeighborElements ()

Create node neighboring elements.

This function is generally useful when, for a numerical method, one looks for a given node to the list of elements that share this node. Once this function is invoked, one can retrieve the list of neighboring elements of any node ([Node::getNeigEl](#))

void getElementNeighborElements ()

Create element neighboring elements.

This function creates for each element the list of elements that share a side with it. Once this function is invoked, one can retrieve the list of neighboring elements of any element (Element::getElementNeighborElement)

void setMaterial (int code, const string & mname)

Associate material to code of element.

Parameters

<code>in</code>	<code>code</code>	Element code for which material is assigned
<code>in</code>	<code>mname</code>	Name of material

void Reorder (size_t m = GRAPH_MEMORY)

Renummer mesh nodes according to reverse Cuthill Mc Kee algorithm.

Parameters

<code>in</code>	<code>m</code>	Memory size needed for matrix graph (default value is GRAPH_MEMORY, see OFELI_Config.h)
-----------------	----------------	---

void Add (size_t num, real_t * x)

Add a node by giving its label and an array containing its coordinates.

Parameters

<code>in</code>	<code>num</code>	Label of node to add
<code>in</code>	<code>x</code>	C-array of node coordinates

void DeleteNode (size_t label)

Remove a node given by its label.

This function does not release the space previously occupied

Parameters

<code>in</code>	<code>label</code>	Label of node to delete
-----------------	--------------------	-------------------------

void DeleteElement (size_t label)

Remove an element given by its label.

This function does not release the space previously occupied

Parameters

<code>in</code>	<code>label</code>	Label of element to delete
-----------------	--------------------	----------------------------

void DeleteSide (size_t label)

Remove a side given by its label.

This function does not release the space previously occupied

Parameters

<code>in</code>	<code>label</code>	Label of side to delete
-----------------	--------------------	-------------------------

void Delete (Node * nd)

Remove a node given by its pointer.

This function does not release the space previously occupied

Parameters

in	<i>nd</i>	Pointer to node to delete
-----------	-----------	---------------------------

void Delete (Element * el)

Remove a node given by its pointer.

This function does not release the space previously occupied

Parameters

in	<i>el</i>	Pointer to element to delete
-----------	-----------	------------------------------

void Delete (Side * sd)

Remove a side given by its pointer.

This function does not release the space previously occupied

Parameters

in	<i>sd</i>	Pointer to side to delete
-----------	-----------	---------------------------

void Delete (Edge * ed)

Remove an edge given by its pointer.

This function does not release the space previously occupied

Parameters

in	<i>ed</i>	Pointer to edge to delete
-----------	-----------	---------------------------

void RenumberNode (size_t n1, size_t n2)

Renumber a node.

Parameters

in	<i>n1</i>	Old label
in	<i>n2</i>	New label

void RenumberElement (size_t n1, size_t n2)

Renumber an element.

Parameters

in	<i>n1</i>	Old label
in	<i>n2</i>	New label

void RenumberSide (size_t n1, size_t n2)

Renumber a side.

Parameters

in	<i>n1</i>	Old label
in	<i>n2</i>	New label

void RenumberEdge (size_t n1, size_t n2)

Renumber an edge.

Parameters

<code>in</code>	<code>n1</code>	Old label
<code>in</code>	<code>n2</code>	New label

void setNodeView (size_t n1, size_t n2)

Set viewing window for nodes.

Parameters

<code>in</code>	<code>n1</code>	First node to view
<code>in</code>	<code>n2</code>	last node to view

void setElementView (size_t n1, size_t n2)

Set viewing window for elements.

Parameters

<code>in</code>	<code>n1</code>	First element to view
<code>in</code>	<code>n2</code>	last element to view

void setSideView (size_t n1, size_t n2)

Set viewing window for sides.

Parameters

<code>in</code>	<code>n1</code>	First side to view
<code>in</code>	<code>n2</code>	last side to view

void setEdgeView (size_t n1, size_t n2)

Set viewing window for edges.

Parameters

<code>in</code>	<code>n1</code>	First edge to view
<code>in</code>	<code>n2</code>	last edge to view

void setList (const std::vector< Node * > & nl)

Initialize list of mesh nodes using the input vector.

Parameters

<code>in</code>	<code>nl</code>	vector instance that contains the list of pointers to nodes
-----------------	-----------------	---

void setList (const std::vector< Element * > & el)

Initialize list of mesh elements using the input vector.

Parameters

<code>in</code>	<code>el</code>	vector instance that contains the list of pointers to elements
-----------------	-----------------	--

void setList (const std::vector< Side * > & sl)

Initialize list of mesh sides using the input vector.

Parameters

in	<i>s/</i>	vector instance that contains the list of pointers to sides
-----------	-----------	---

void Rescale (real_t sx, real_t sy = 0., real_t sz = 0.)

Rescale mesh by multiplying node coordinates by constants.

This function can be used e.g. for changing coordinate units

Parameters

in	<i>sx</i>	Factor to multiply by x coordinates
in	<i>sy</i>	Factor to multiply by y coordinates [Default: <i>sx</i>]
in	<i>sz</i>	Factor to multiply by z coordinates [Default: <i>sx</i>]

size_t getNbBoundarySides () const

Return number of boundary sides.

This function is valid if member function **getAllSides** or **getBoundarySides** has been invoked before

size_t getNbInternalSides () const

Return number of internal sides.

This function is valid if member functions **getAllSides** and **createInternalSideList** have been invoked before

void AddMidNodes (int g = 0)

Add mid-side nodes.

This is function is valid for triangles only

Parameters

in	<i>g</i>	Option to say of barycentre node is to be added (>0) or not (=0)
-----------	----------	--

void set (Node * nd)

Replace node in the mesh.

If the node label exists already, the existing node pointer will be replaced by the current one. If not, an error message is displayed.

Parameters

in	<i>nd</i>	Pointer to node
-----------	-----------	-----------------

void set (Element * el)

Replace element in the mesh.

If the element label exists already, the existing element pointer will be replaced by the current one. If not, an error message is displayed.

Parameters

in	<i>el</i>	Pointer to element
-----------	-----------	--------------------

void set (Side * sd)

Choose side in the mesh.

If the side label exists already, the existing side pointer will be replaced by the current one. If not, an error message is displayed.

Parameters

<i>in</i>	<i>sd</i>	Pointer to side
-----------	-----------	-----------------

bool NodesAreDOF () const

Return information about DOF type.

Returns

true if DOF are supported by nodes, false otherwise

bool SidesAreDOF () const

Return information about DOF type.

Returns

true if DOF are supported by sides, false otherwise

bool EdgesAreDOF () const

Return information about DOF type.

Returns

true if DOF are supported by edges, false otherwise

bool ElementsAreDOF () const

Return information about DOF type.

Returns

true if DOF are supported by elements, false otherwise

void put (const string & mesh_file) const

Write mesh data on file.

Parameters

<i>in</i>	<i>mesh_file</i>	Mesh file name
-----------	------------------	----------------

void save (const string & mesh_file) const

Write mesh data on file in various formats.

File format depends on the extension in file name

Parameters

in	<i>mesh_file</i>	Mesh file name If the extension is '.m', the output file is an OFF- LI file If the extension is '.gpl', the output file is a Gnuplot file If the extension is '.msh' or '.geo', the output file is a Gmsh file If the extension is '.vtk', the output file is a VTK file
-----------	------------------	--

void Bput (const string & mesh_file) const

Write mesh data on a binary file.

Parameters

in	<i>mesh_file</i>	Mesh file name
-----------	------------------	----------------

void getList (vector< Node * > & nl) const

Fill vector *nl* with list of pointers to nodes.

Parameters

out	<i>nl</i>	Instance of class vector that contain on output the list
------------	-----------	--

void getList (vector< Element * > & el) const

Fill vector *el* with list of pointers to elements.

Parameters

out	<i>el</i>	Instance of class vector that contain on output the list
------------	-----------	--

void getList (vector< Side * > & sl) const

Fill vector *sl* with list of pointers to sides.

Parameters

out	<i>sl</i>	Instance of class vector that contain on output the list
------------	-----------	--

size_t getNodeLabel (size_t i) const

Return label of *i*-th node.

Parameters

in	<i>i</i>	Node index
-----------	----------	----------------------------

size_t getElementLabel (size_t i) const

Return label of *i*-th element.

Parameters

in	<i>i</i>	Element index
-----------	----------	-------------------------------

size_t getSideLabel (size_t i) const

Return label of *i*-th side.

Parameters

<code>in</code>	<code>i</code>	Side index
-----------------	----------------	----------------------------

size_t getEdgeLabel (size_t i) const

Return label of i-th edge.

Parameters

<code>in</code>	<code>i</code>	Edge index
-----------------	----------------	----------------------------

Element* getActiveElement () const

Return pointer to current element and move to next one (Non constant version)

This function returns pointer to the current element only if this one is active. Otherwise it goes to the next active element (To be used when adaptive meshing is involved)

7.66.4 Friends And Related Function Documentation

void Refine (Mesh & in_mesh, Mesh & out_mesh) [friend]

Refine mesh. Subdivide each triangle into 4 subtriangles. This member function is valid for 2-D triangular meshes only.

Parameters

<code>in</code>	<code>in_mesh</code>	Input mesh
<code>out</code>	<code>out_mesh</code>	Output mesh

7.67 MeshAdapt Class Reference

To adapt mesh in function of given solution.

Public Member Functions

- [MeshAdapt](#) ()
Default constructor.
- [MeshAdapt](#) (Mesh &ms)
Constructor using initial mesh.
- [MeshAdapt](#) (Domain &dom)
Constructor using a reference to class [Domain](#).
- [~MeshAdapt](#) ()
Destructor.
- [Domain](#) & [getDomain](#) () const
Get reference to [Domain](#) instance.
- [Mesh](#) & [getMesh](#) () const
Get reference to current mesh.
- void [set](#) (Domain &dom)
Set reference to [Domain](#) instance.
- void [set](#) (Mesh &ms)
Set reference to [Mesh](#) instance.
- void [setSolution](#) (const Vect< real_t > &u)

- Define label of node.*
- void [setJacobi](#) (int n)
 - Set number of Jacobi iterations for smoothing.*
- void [setSmooth](#) (int n)
 - Set number of smoothing iterations.*
- void [setVerbosity](#) (int verb)
 - Set verbosity parameter.*
- void [AbsoluteError](#) ()
 - Metric is constructed with absolute error.*
- void [RelativeError](#) ()
 - Metric is constructed with relative error.*
- void [setError](#) (real_t err)
 - Set error threshold for adaption.*
- void [setHMin](#) (real_t h)
 - Set minimal mesh size.*
- void [setHMax](#) (real_t h)
 - Set maximal mesh size.*
- void [setHMinAnisotropy](#) (real_t h)
 - Set minimal mesh size and set anisotropy.*
- void [setRelaxation](#) (real_t omega)
 - Set relaxation parameter for smoothing.*
- void [setAnisotropic](#) ()
 - Set that adapted mesh construction is anisotropic.*
- void [MaxAnisotropy](#) (real_t a)
 - Set maximum ratio of anisotropy.*
- void [setMaxSubdiv](#) (real_t s)
 - Change the metric such that the maximal subdivision of a background's edge is bounded by the given number (always limited by 10)*
- void [setMaxNbVertices](#) (size_t n)
 - Set maximum number of vertices.*
- void [setRatio](#) (real_t r)
 - Set ratio for a smoothing of the metric.*
- void [setNoScaling](#) ()
 - Do not scale solution before metric computation.*
- void [setNoKeep](#) ()
 - Do not keep old vertices.*
- void [setHessian](#) ()
 - set computation of the Hessian*
- void [setOutputMesh](#) (string file)
 - Create mesh output file.*
- void [setGeoFile](#) (string file)
 - Set Geometry file.*
- void [setGeoError](#) (real_t e)
 - Set error on geometry.*
- void [setBackgroundMesh](#) (string bgm)
 - Set background mesh.*
- void [SplitBoundaryEdges](#) ()

- void [CreateMetricFile](#) (string mf)
Create a metric file.
- void [setMetricFile](#) (string mf)
Set Metric file.
- void [getSolutionMbb](#) (string mbb)
Set solution defined on background mesh for metric construction.
- void [getSolutionMBB](#) (string mBB)
Set solution defined on background mesh for metric construction.
- void [getSolutionbb](#) (string rbb)
Read solution defined on the background mesh in bb file.
- void [getSolutionBB](#) (string rBB)
Read solution defined on the background mesh in BB file.
- void [getSolution](#) (Vect< real_t > &u, int is=1)
Get the interpolated solution on the new mesh.
- void [getInterpolatedSolutionbb](#) ()
Write the file of interpolation of the solutions in bb file.
- void [getInterpolatedSolutionBB](#) ()
Write the file of interpolation of the solutions in BB file.
- void [setTheta](#) (real_t theta)
Set angular limit for a corner (in degrees)
- void [Split](#) ()
Split triangles into 4 triangles.
- void [saveMbb](#) (string file, const Vect< real_t > &u)
Save a solution in metric file.
- int [run](#) ()
Run adaption process.

7.67.1 Detailed Description

To adapt mesh in function of given solution.

Class [MeshAdapt](#) enables modifying mesh according to a solution vector defining at nodes. It concerns 2-D triangular meshes only.

Remarks

Class [MeshAdapt](#) is mainly based on the software 'Bamg' developed by F. Hecht, Universite Pierre et Marie Curie, Paris. We warmly thank him for accepting incorporation of Bamg in the [OFELI](#) package

7.67.2 Constructor & Destructor Documentation

MeshAdapt (Mesh & ms)

Constructor using initial mesh.

Parameters

<code>in</code>	<code>ms</code>	Reference to initial mesh
-----------------	-----------------	---------------------------

MeshAdapt (Domain & dom)

Constructor using a reference to class [Domain](#).

Parameters

<i>in</i>	<i>dom</i>	Reference to Domain class
-----------	------------	---

7.67.3 Member Function Documentation

void setRelaxation (*real_t* omega)

Set relaxation parameter for smoothing.
Default value for relaxation parameter is 1.8

void setMaxNbVertices (*size_t* n)

Set maximum number of vertices.
Default value is 500000

void setRatio (*real_t* r)

Set ratio for a smoothing of the metric.
Parameters

<i>in</i>	<i>r</i>	Ratio value.
-----------	----------	--------------

Note

If *r* is 0 then no smoothing is performed, if *r* lies in [1.1,10] then the smoothing changes the metric such that the largest geometrical progression (speed of mesh size variation in mesh is bounded by *r* (by default no smoothing)

void setNoScaling ()

Do not scale solution before metric computation.
By default, solution is scaled (between 0 and 1)

void setNoKeep ()

Do not keep old vertices.
By default, old vertices are kept

void getSolutionbb (*string* rbb)

Read solution defined on the background mesh in bb file.
Solution is interpolated on created mesh

void getSolutionBB (*string* rBB)

Read solution defined on the background mesh in BB file.
Solution is interpolated on created mesh

void getSolution (*Vect< real_t > & u*, *int* is = 1)

Get the interpolated solution on the new mesh.
The solution must have been saved on an output bb file

Parameters

out	<i>u</i>	Vector that contains on output the obtained solutions. This vector is resized before being initialized
in	<i>is</i>	[Default: 1]

void setTheta (real_t theta)

Set angular limit for a corner (in degrees)

The angle is defined from 2 normals of 2 consecutive edges

void saveMbb (string file, const Vect< real_t > & u)

Save a solution in metric file.

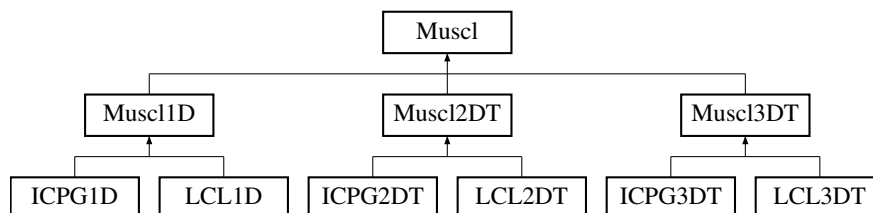
Parameters

in	<i>file</i>	File name where the metric is stored
in	<i>u</i>	Solution vector to store

7.68 Muscl Class Reference

Parent class for hyperbolic solvers with Muscl scheme.

Inheritance diagram for Muscl:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `Muscl` (`Mesh` &m)
Constructor using mesh instance.
- virtual `~Muscl` ()
Destructor.
- void `setTimeStep` (`real_t` dt)
Assign time step value.
- `real_t` `getTimeStep` () const
Return time step value.
- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getCFL` () const
Return CFL value.
- void `setReferenceLength` (`real_t` dx)
Assign reference length value.
- `real_t` `getReferenceLength` () const
Return reference length.
- `Mesh` & `getMesh` () const
Return reference to `Mesh` instance.
- void `setVerbose` (int v)
Set verbosity parameter.
- bool `setReconstruction` (const `Vect`< `real_t` > &U, `Vect`< `real_t` > &LU, `Vect`< `real_t` > &RU, size_t dof)
Function to reconstruct by the `Muscl` method.
- void `setMethod` (const `Method` &s)
Choose a flux solver.
- void `setSolidZoneCode` (int c)
Choose a code for solid zone.
- bool `getSolidZone` () const
Return flag for presence of solid zones.
- int `getSolidZoneCode` () const
Return code of solid zone, 0 if this one is not present.
- void `setLimiter` (`Limiter` l)
Choose a flux limiter.

7.68.1 Detailed Description

Parent class for hyperbolic solvers with Muscl scheme.

Everything here is common for both 2D and 3D muscl methods ! Virtual functions are implemented in Muscl2D and Muscl3D classes

7.68.2 Member Enumeration Documentation

enum Method

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method
MULTI_SLOPE_Q_METHOD Multislope Q method
MULTI_SLOPE_M_METHOD Multislope M method
M_FORCE_WORD for compatabity reasons, gcc may return a warning anyway

enum Limiter

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatabity reason, gcc may return a warning anyway

enum SolverType

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver
VFROE_SOLVER Finite Volume Roe solver
LF_SOLVER LF solver
RUSANOV_SOLVER Rusanov solver
HLL_SOLVER HLL solver
HLLC_SOLVER HLLC solver
MAX_SOLVER Max solver
FORCE_WORD for compatabity reason, gcc may return a warning anyway

7.68.3 Member Function Documentation

void setTimeStep (real_t dt)

Assign time step value.

Parameters

<code>in</code>	<code>dt</code>	Time step value
-----------------	-----------------	-----------------

void setCFL (real_t CFL)

Assign CFL value.

Parameters

<code>in</code>	<code>CFL</code>	Value of CFL
-----------------	------------------	--------------

void setReferenceLength (real_t dx)

Assign reference length value.

Parameters

<code>in</code>	<code>dx</code>	Value of reference length
-----------------	-----------------	---------------------------

void setVerbose (int v)

Set verbosity parameter.

Parameters

<code>in</code>	<code>v</code>	Value of verbosity parameter
-----------------	----------------	------------------------------

bool setReconstruction (const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t > & RU, size_t dof)

Function to reconstruct by the [Muscl](#) method.

Parameters

<code>in</code>	<code>U</code>	Field to reconstruct
<code>out</code>	<code>LU</code>	Left gradient vector
<code>out</code>	<code>RU</code>	Right gradient vector
<code>in</code>	<code>dof</code>	Label of dof to reconstruct

void setMethod (const Method & s)

Choose a flux solver.

Parameters

<code>in</code>	<code>s</code>	Solver to choose
-----------------	----------------	------------------

void setLimiter (Limiter l)

Choose a flux limiter.

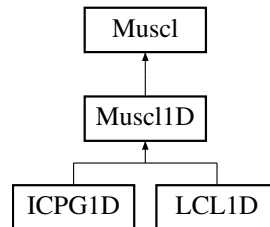
Parameters

<code>in</code>	<code>l</code>	Limiter to choose
-----------------	----------------	-------------------

7.69 Muscl1D Class Reference

Class for 1-D hyperbolic solvers with [Muscl](#) scheme.

Inheritance diagram for Muscl1D:



Public Types

- enum [Method](#) {
[FIRST_ORDER_METHOD](#) = 0,
[MULTI_SLOPE_Q_METHOD](#) = 1,
[MULTI_SLOPE_M_METHOD](#) = 2,
[M_FORCE_WORD](#) = 0xFFFFFFFF }

Enumeration for flux choice.

- enum [Limiter](#) {
[MINMOD_LIMITER](#) = 0,
[VANLEER_LIMITER](#) = 1,
[SUPERBEE_LIMITER](#) = 2,
[VANALBADA_LIMITER](#) = 3,
[MAX_LIMITER](#) = 4,
[L_FORCE_WORD](#) = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum [SolverType](#) {
[ROE_SOLVER](#) = 0,
[VFROE_SOLVER](#) = 1,
[LF_SOLVER](#) = 2,
[RUSANOV_SOLVER](#) = 3,
[HLL_SOLVER](#) = 4,
[HLLC_SOLVER](#) = 5,
[MAX_SOLVER](#) = 6,
[FORCE_WORD](#) = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- [Muscl1D](#) ([Mesh](#) &m)
Constructor using mesh instance.
- [~Muscl1D](#) ()
Destructor.
- [real_t](#) [getMeanLength](#) () const
Return mean length.
- [real_t](#) [getMaximumLength](#) () const
Return maximal length.
- [real_t](#) [getMinimumLength](#) () const

- Return mimal length.*
- `real_t getTauLim ()` const
- Return mean length.*
- `void print_mesh_stat ()`
- Output mesh information.*
- `void setTimeStep (real_t dt)`
- Assign time step value.*
- `real_t getTimeStep ()` const
- Return time step value.*
- `void setCFL (real_t CFL)`
- Assign CFL value.*
- `real_t getCFL ()` const
- Return CFL value.*
- `void setReferenceLength (real_t dx)`
- Assign reference length value.*
- `real_t getReferenceLength ()` const
- Return reference length.*
- `Mesh & getMesh ()` const
- Return reference to [Mesh](#) instance.*
- `void setVerbose (int v)`
- Set verbosity parameter.*
- `bool setReconstruction (const Vect< real_t > &U, Vect< real_t > &LU, Vect< real_t > &RU, size_t dof)`
- Function to reconstruct by the [Muscl](#) method.*
- `void setMethod (const Method &s)`
- Choose a flux solver.*
- `void setSolidZoneCode (int c)`
- Choose a code for solid zone.*
- `bool getSolidZone ()` const
- Return flag for presence of solid zones.*
- `int getSolidZoneCode ()` const
- Return code of solid zone, 0 if this one is not present.*
- `void setLimiter (Limiter l)`
- Choose a flux limiter.*

7.69.1 Detailed Description

Class for 1-D hyperbolic solvers with [Muscl](#) scheme.

7.69.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

- FIRST_ORDER_METHOD** First Order upwind method
- MULTI_SLOPE_Q_METHOD** Multislope Q method
- MULTI_SLOPE_M_METHOD** Multislope M method
- M_FORCE_WORD** for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter

VANLEER_LIMITER Van Leer limiter

SUPERBEE_LIMITER Superbee limiter

VANALBADA_LIMITER Van Albada limiter

MAX_LIMITER Max limiter

L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver

VFROE_SOLVER Finite Volume Roe solver

LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.69.3 Member Function Documentation

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	<i>dx</i>	Value of reference length
----	-----------	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	<i>v</i>	Value of verbosity parameter
----	----------	------------------------------

bool setReconstruction (const Vect< real.t > & U, Vect< real.t > & LU, Vect< real.t > & RU, size.t dof) [inherited]

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	<i>s</i>	Solver to choose
----	----------	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

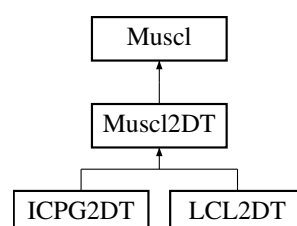
Parameters

in	<i>l</i>	Limiter to choose
----	----------	-------------------

7.70 Muscl2DT Class Reference

Class for 2-D hyperbolic solvers with [Muscl](#) scheme.

Inheritance diagram for Muscl2DT:



Public Types

- enum `Method` {
`FIRST_ORDER_METHOD` = 0,
`MULTI_SLOPE_Q_METHOD` = 1,
`MULTI_SLOPE_M_METHOD` = 2,
`M_FORCE_WORD` = 0xFFFFFFFF }

Enumeration for flux choice.

- enum `Limiter` {
`MINMOD_LIMITER` = 0,
`VANLEER_LIMITER` = 1,
`SUPERBEE_LIMITER` = 2,
`VANALBADA_LIMITER` = 3,
`MAX_LIMITER` = 4,
`L_FORCE_WORD` = 0xFFFFFFFF }

Enumeration of flux limiting methods.

- enum `SolverType` {
`ROE_SOLVER` = 0,
`VFROE_SOLVER` = 1,
`LF_SOLVER` = 2,
`RUSANOV_SOLVER` = 3,
`HLL_SOLVER` = 4,
`HLLC_SOLVER` = 5,
`MAX_SOLVER` = 6,
`FORCE_WORD` = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- `Muscl2DT` (`Mesh` &m)
Constructor using mesh.
- `~Muscl2DT` ()
Destructor.
- bool `setReconstruction` (const `Vect`< `real_t` > &U, `Vect`< `real_t` > &LU, `Vect`< `real_t` > &RU, size_t dof)
Function to reconstruct by the `Muscl` method.
- void `setTimeStep` (`real_t` dt)
Assign time step value.
- `real_t` `getTimeStep` () const
Return time step value.
- void `setCFL` (`real_t` CFL)
Assign CFL value.
- `real_t` `getCFL` () const
Return CFL value.
- void `setReferenceLength` (`real_t` dx)
Assign reference length value.
- `real_t` `getReferenceLength` () const
Return reference length.
- `Mesh` & `getMesh` () const
Return reference to `Mesh` instance.

- void `setVerbose` (int v)
Set verbosity parameter.
- void `setMethod` (const `Method` &s)
Choose a flux solver.
- void `setSolidZoneCode` (int c)
Choose a code for solid zone.
- bool `getSolidZone` () const
Return flag for presence of solid zones.
- int `getSolidZoneCode` () const
Return code of solid zone, 0 if this one is not present.
- void `setLimiter` (`Limiter` l)
Choose a flux limiter.

Protected Member Functions

- void `Initialize` ()
Construction of normals to sides.

7.70.1 Detailed Description

Class for 2-D hyperbolic solvers with `Muscl` scheme.

7.70.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

FIRST_ORDER_METHOD First Order upwind method
MULTI_SLOPE_Q_METHOD Multislope Q method
MULTI_SLOPE_M_METHOD Multislope M method
M_FORCE_WORD for compatibility reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

MINMOD_LIMITER MinMod limiter
VANLEER_LIMITER Van Leer limiter
SUPERBEE_LIMITER Superbee limiter
VANALBADA_LIMITER Van Albada limiter
MAX_LIMITER Max limiter
L_FORCE_WORD for compatibility reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver

VFROE_SOLVER Finite Volume Roe solver

LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatibility reason, gcc may return a warning anyway

7.70.3 Member Function Documentation

bool setReconstruction (const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t > & RU, size_t dof)

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void Initialize () [protected]

Construction of normals to sides.

Convention: for a given side, `getPtrElement(1)` is the left element and `getPtrElement(2)` is the right element. The normal goes from left to right. For boundary sides, the normal points outward.

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	dx	Value of reference length
----	----	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	v	Value of verbosity parameter
----	---	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	s	Solver to choose
----	---	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

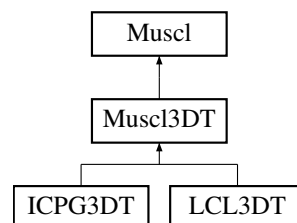
Parameters

in	/	Limiter to choose
----	---	-------------------

7.71 Muscl3DT Class Reference

Class for 3-D hyperbolic solvers with [Muscl](#) scheme using tetrahedra.

Inheritance diagram for Muscl3DT:



Public Types

- enum [Method](#) {
[FIRST_ORDER_METHOD](#) = 0,
[MULTI_SLOPE_Q_METHOD](#) = 1,
[MULTI_SLOPE_M_METHOD](#) = 2,
[M_FORCE_WORD](#) = 0xFFFFFFFF }

Enumeration for flux choice.

- enum [Limiter](#) {
[MINMOD_LIMITER](#) = 0,
[VANLEER_LIMITER](#) = 1,
[SUPERBEE_LIMITER](#) = 2,
[VANALBADA_LIMITER](#) = 3,
[MAX_LIMITER](#) = 4,

```
L_FORCE_WORD = 0xFFFFFFFF }
```

Enumeration of flux limiting methods.

- enum SolverType {
 ROE_SOLVER = 0,
 VFROE_SOLVER = 1,
 LF_SOLVER = 2,
 RUSANOV_SOLVER = 3,
 HLL_SOLVER = 4,
 HLLC_SOLVER = 5,
 MAX_SOLVER = 6,
 FORCE_WORD = 0xFFFFFFFF }

Enumeration of various solvers for the Riemann problem.

Public Member Functions

- Muscl3DT (Mesh &m)
Constructor using mesh.
- ~Muscl3DT ()
Destructor.
- bool setReconstruction (const Vect< real_t > &U, Vect< real_t > &LU, Vect< real_t > &RU, size_t dof)
Function to reconstruct by the Muscl method.
- real_t getMinimumFaceArea () const
Return minimum area of faces in the mesh.
- real_t getMinimumElementVolume () const
Return minimum volume of elements in the mesh.
- real_t getMaximumFaceArea () const
Return maximum area of faces in the mesh.
- real_t getMaximumElementVolume () const
Return maximum volume of elements in the mesh.
- real_t getMeanFaceArea () const
Return mean area of faces in the mesh.
- real_t getMeanElementVolume () const
Return mean volume of elements in the mesh.
- real_t getMinimumEdgeLength () const
Return minimum length of edges in the mesh.
- real_t getMinimumVolumebyArea () const
Return minimum volume by area in the mesh.
- real_t getMaximumEdgeLength () const
Return maximum length of edges in the mesh.
- real_t getTauLim () const
Return value of tau lim.
- real_t getComega () const
Return value of Comega.
- void setbetalim (real_t bl)
Assign value of beta lim.
- void setTimeStep (real_t dt)
Assign time step value.

- `real_t getTimeStep ()` const
Return time step value.
- `void setCFL (real_t CFL)`
Assign CFL value.
- `real_t getCFL ()` const
Return CFL value.
- `void setReferenceLength (real_t dx)`
Assign reference length value.
- `real_t getReferenceLength ()` const
Return reference length.
- `Mesh & getMesh ()` const
Return reference to [Mesh](#) instance.
- `void setVerbose (int v)`
Set verbosity parameter.
- `void setMethod (const Method &s)`
Choose a flux solver.
- `void setSolidZoneCode (int c)`
Choose a code for solid zone.
- `bool getSolidZone ()` const
Return flag for presence of solid zones.
- `int getSolidZoneCode ()` const
Return code of solid zone, 0 if this one is not present.
- `void setLimiter (Limiter l)`
Choose a flux limiter.

7.71.1 Detailed Description

Class for 3-D hyperbolic solvers with [Muscl](#) scheme using tetrahedra.

7.71.2 Member Enumeration Documentation

enum Method [inherited]

Enumeration for flux choice.

Enumerator

- FIRST_ORDER_METHOD** First Order upwind method
- MULTI_SLOPE_Q_METHOD** Multislope Q method
- MULTI_SLOPE_M_METHOD** Multislope M method
- M_FORCE_WORD** for compatability reasons, gcc may return a warning anyway

enum Limiter [inherited]

Enumeration of flux limiting methods.

Enumerator

- MINMOD_LIMITER** MinMod limiter
- VANLEER_LIMITER** Van Leer limiter

SUPERBEE_LIMITER Superbee limiter

VANALBADA_LIMITER Van Albada limiter

MAX_LIMITER Max limiter

L_FORCE_WORD for compatabity reason, gcc may return a warning anyway

enum SolverType [inherited]

Enumeration of various solvers for the Riemann problem.

Enumerator

ROE_SOLVER Roe solver

VFROE_SOLVER Finite Volume Roe solver

LF_SOLVER LF solver

RUSANOV_SOLVER Rusanov solver

HLL_SOLVER HLL solver

HLLC_SOLVER HLLC solver

MAX_SOLVER Max solver

FORCE_WORD for compatabity reason, gcc may return a warning anyway

7.71.3 Member Function Documentation

bool setReconstruction (const Vect< real_t > & U, Vect< real_t > & LU, Vect< real_t > & RU, size_t dof)

Function to reconstruct by the [Muscl](#) method.

Parameters

in	<i>U</i>	Field to reconstruct
out	<i>LU</i>	Left gradient vector
out	<i>RU</i>	Right gradient vector
in	<i>dof</i>	Label of dof to reconstruct

void setTimeStep (real_t dt) [inherited]

Assign time step value.

Parameters

in	<i>dt</i>	Time step value
----	-----------	-----------------

void setCFL (real_t CFL) [inherited]

Assign CFL value.

Parameters

in	<i>CFL</i>	Value of CFL
----	------------	--------------

void setReferenceLength (real_t dx) [inherited]

Assign reference length value.

Parameters

in	dx	Value of reference length
----	----	---------------------------

void setVerbose (int v) [inherited]

Set verbosity parameter.

Parameters

in	v	Value of verbosity parameter
----	---	------------------------------

void setMethod (const Method & s) [inherited]

Choose a flux solver.

Parameters

in	s	Solver to choose
----	---	------------------

void setLimiter (Limiter l) [inherited]

Choose a flux limiter.

Parameters

in	/	Limiter to choose
----	---	-------------------

7.72 Node Class Reference

To describe a node.

Public Member Functions

- [Node](#) ()
Default constructor.
- [Node](#) (size_t label, const [Point](#)< double > &x)
Constructor with label and coordinates.
- [Node](#) (const [Node](#) &node)
Copy Constructor.
- [~Node](#) ()
Destructor.
- void [setLabel](#) (size_t label)
Define label of node.
- void [setNbDOF](#) (size_t n)
Define number of DOF.
- void [setFirstDOF](#) (size_t n)
Define First DOF.
- void [setCode](#) (size_t dof, int code)
Define code for a given DOF of node.
- void [setCode](#) (const vector< int > &code)
Define codes for all node DOFs.
- void [setCode](#) (int *code)

- Define codes for all node DOFs.*

 - void `setCode` (const string &exp, int code, size_t dof=1)
Define code by a boolean algebraic expression invoking node coordinates.
 - void `setCoord` (size_t i, double x)
Set i-th coordinate.
 - void `DOF` (size_t i, size_t dof)
Define label of DOF.
 - void `setDOF` (size_t &first_dof, size_t nb_dof)
Define number of DOF.
 - void `setOnBoundary` ()
Set node as boundary node.
 - size_t `n` () const
Return label of node.
 - size_t `getNbDOF` () const
Return number of degrees of freedom (DOF)
 - int `getCode` (size_t dof=1) const
Return code for a given DOF of node.
 - double `getCoord` (size_t i) const
Return i-th coordinate of node. i = 1..3.
 - `Point< double >` `getCoord` () const
Return coordinates of node.
 - double `getX` () const
Return x-coordinate of node.
 - double `getY` () const
Return y-coordinate of node.
 - double `getZ` () const
Return z-coordinate of node.
 - `Point< double >` `getXYZ` () const
Return coordinates of node.
 - size_t `getDOF` (size_t i) const
Return label of i-th dof.
 - size_t `getNbNeigEl` () const
Return number of neighbor elements.
 - `Element *` `getNeigEl` (size_t i) const
Return i-th neighbor element.
 - size_t `getFirstDOF` () const
Return label of first DOF of node.
 - bool `isOnBoundary` () const
Say if node is a boundary node.
 - void `Add` (`Element *el`)
Add element pointed by el as neighbor element to node.
 - void `setLevel` (int level)
Assign a level to current node.
 - int `getLevel` () const
Return node level `Node` level decreases when element is refined (starting from 0). If the level is 0, then the element has no parents.

7.72.1 Detailed Description

To describe a node.

A node is characterized by its label, its coordinates, its number of degrees of freedom (DOF) and codes that are associated to each DOF.

Remarks

Once the mesh is constructed, information on neighboring elements of node can be retrieved (see appropriate member functions). However, the member function `getNodeNeighborElements` of [Mesh](#) must have been called before. If this is not the case, the program crashes down since no preliminary checking is done for efficiency reasons.

7.72.2 Constructor & Destructor Documentation

Node ()

Default constructor.

Initialize data to zero

Node (`size_t` label, `const Point< double > &` x)

Constructor with label and coordinates.

Parameters

<code>in</code>	<i>label</i>	Label of node
<code>in</code>	<i>x</i>	Node coordinates

7.72.3 Member Function Documentation

void setCode (`size_t` dof, `int` code)

Define code for a given DOF of node.

Parameters

<code>in</code>	<i>dof</i>	DOF index
<code>in</code>	<i>code</i>	Code to assign to DOF

void setCode (`const vector< int > &` code)

Define codes for all node DOFs.

Parameters

<code>in</code>	<i>code</i>	vector instance that contains code for each DOF of current node
-----------------	-------------	---

void setCode (`int *` code)

Define codes for all node DOFs.

Parameters

<code>in</code>	<i>code</i>	C-array that contains code for each DOF of current node
-----------------	-------------	---

void setCode (`const string &` exp, `int` code, `size_t` dof = 1)

Define code by a boolean algebraic expression invoking node coordinates.

Parameters

in	<i>exp</i>	Boolean algebraic expression as required by <code>fparser</code>
in	<i>code</i>	Code to assign to node if the algebraic expression is true
in	<i>dof</i>	Degree of Freedom for which code is assigned [Default: 1]

void setCoord (size_t i, double x)

Set i-th coordinate.

Parameters

in	<i>i</i>	Coordinate index (1..3)
in	<i>x</i>	Coordinate value

void DOF (size_t i, size_t dof)

Define label of DOF.

Parameters

in	<i>i</i>	DOF index
in	<i>dof</i>	Label of DOF

void setDOF (size_t & first_dof, size_t nb_dof)

Define number of DOF.

Parameters

in,out	<i>first_dof</i>	Label of the first DOF in input that is actualized
in	<i>nb_dof</i>	Number of DOF

void setOnBoundary ()

Set node as boundary node.

This function is mostly internally used (Especially in class [Mesh](#))

int getCode (size_t dof = 1) const

Return code for a given DOF of node.

Parameters

in	<i>dof</i>	label of degree of freedom for which code is to be returned. Default value is 1.
-----------	------------	--

Point<double> getCoord () const

Return coordinates of node.

Return value is an instance of class [Point](#)

Point<double> getXYZ () const

Return coordinates of node.

Return value is an instance of class [Point](#)

size_t getNbNeigEl () const

Return number of neighbor elements.

Neighbor elements are those that share node. Note that the returned information is valid only if the [Mesh](#) member function **getNodeNeighborElements()** has been invoked before

Element* getNeigEl (size_t i) const

Return i-th neighbor element.

Note that the returned information is valid only if the [Mesh](#) member function **getNodeNeighborElements()** has been invoked before

bool isOnBoundary () const

Say if node is a boundary node.

Note this information is available only if boundary sides (and nodes) were determined (See class [Mesh](#)).

void setLevel (int level)

Assign a level to current node.

This member function is useful for mesh adaption.

Default node's level is zero

7.73 NodeList Class Reference

Class to construct a list of nodes having some common properties.

Public Member Functions

- [NodeList](#) ([Mesh](#) &ms)
Constructor using a [Mesh](#) instance.
- [~NodeList](#) ()
Destructor.
- void [selectCode](#) (int code, int dof=1)
Select nodes having a given code for a given degree of freedom.
- void [unselectCode](#) (int code, int dof=1)
Unselect nodes having a given code for a given degree of freedom.
- void [selectCoordinate](#) ([real_t](#) x, [real_t](#) y=ANY, [real_t](#) z=ANY)
Select nodes having given coordinates.
- size_t [getNbNodes](#) () const
Return number of selected nodes.
- void [top](#) ()
Reset list of nodes at its top position (Non constant version)
- void [top](#) () const
Reset list of nodes at its top position (Constant version)
- [Node](#) * [get](#) ()
Return pointer to current node and move to next one (Non constant version)
- [Node](#) * [get](#) () const
Return pointer to current node and move to next one (Constant version)

7.73.1 Detailed Description

Class to construct a list of nodes having some common properties.

This class enables choosing multiple selection criteria by using function `select...`. However, the intersection of these properties must be empty.

7.73.2 Member Function Documentation

void selectCode (int code, int dof = 1)

Select nodes having a given code for a given degree of freedom.

Parameters

in	<i>code</i>	Code that nodes share
in	<i>dof</i>	Degree of Freedom label [Default: 1]

void unselectCode (int code, int dof = 1)

Unselect nodes having a given code for a given degree of freedom.

Parameters

in	<i>code</i>	Code of nodes to exclude
in	<i>dof</i>	Degree of Freedom label [Default: 1]

void selectCoordinate (real_t x, real_t y = ANY, real_t z = ANY)

Select nodes having given coordinates.

Parameters

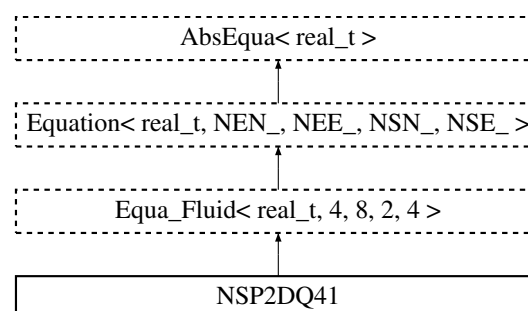
in	<i>x</i>	x-coordinate that share the selected nodes
in	<i>y</i>	y-coordinate that share the selected nodes [Default: ANY]
in	<i>z</i>	z-coordinate that share the selected nodes [Default: ANY]

Coordinates can be assigned the value ANY. This means that any coordinate value is accepted. For instance, to select all nodes with $x=0$, use `selectCoordinate(0.,ANY,ANY)`;

7.74 NSP2DQ41 Class Reference

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using Q_1/P_0 element and a penalty formulation for the incompressibility condition.

Inheritance diagram for NSP2DQ41:



Public Member Functions

- [NSP2DQ41](#) ()
Default Constructor.
- [NSP2DQ41](#) (const [Element](#) *el)
Constructor using [Element](#) data.
- [NSP2DQ41](#) (const [Side](#) *sd)
Constructor using [Side](#) data.
- [NSP2DQ41](#) (const [Element](#) *el, const [Vect](#)< [real_t](#) > &u, const [real_t](#) &time=0.)
Constructor using element and previous time data.
- [NSP2DQ41](#) (const [Side](#) *sd, const [Vect](#)< [real_t](#) > &u, const [real_t](#) &time=0.)
Constructor using side and previous time data.
- [~NSP2DQ41](#) ()
Destructor.
- void [Viscosity](#) ([real_t](#) visc)
Define constant viscosity.
- void [Density](#) ([real_t](#) dens)
Define constant density.
- void [LMass](#) ([real_t](#) coef=1.)
Add element lumped mass contribution to matrix after multiplication by coef [Default: 1].
- void [Mass](#) ([real_t](#) coef=1.)
Add element consistent mass contribution to matrix after multiplication by coef [Default: 1].
- void [Viscous](#) ([real_t](#) coef=1.)
Add element viscous contribution to matrix after multiplication by coef [Default: 1].
- void [RHS_Viscous](#) ([real_t](#) coef=1.)
Add element viscous contribution to right-hand side after multiplication by coef [Default: 1].
- void [Penal](#) ([real_t](#) coef=1.)
Add element penalty contribution to matrix after multiplication by coef [Default: 1].
- void [LHS1_Convection](#) ([real_t](#) coef=1.)
Add convection contribution to left-hand side after multiplication by coef [Default: 1].
- void [LHS2_Convection](#) ([real_t](#) coef=1.)
Add convection contribution to left-hand side after multiplication by coef [Default: 1].
- void [RHS_Convection](#) ([real_t](#) coef=1.)
Add convection contribution to right-hand side after multiplication by coef [Default: 1].
- void [BodyRHS](#) ([UserData](#)< [real_t](#) > &ud)
Add body right-hand side term to right-hand side.
- void [BoundaryRHS](#) ([UserData](#)< [real_t](#) > &ud)
Add boundary right-hand side term to right-hand side.
- void [Periodic](#) ([real_t](#) coef=1.e20)
Add contribution of periodic boundary condition (by a penalty technique).
- [real_t](#) [Pressure](#) ([real_t](#) coef=1.)
Calculate element pressure by penalization after multiplication by coef [Default: 1].
- void [updateBC](#) (const [Element](#) &el, const [Vect](#)< [real_t](#) > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [updateBC](#) (const [Vect](#)< [real_t](#) > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void [DiagBC](#) (int dof.type=NODE.DOF, int dof=0)

- Update element matrix to impose bc by diagonalization technique.*

 - void `LocalNodeVector` (`Vect< real_t > &b`)
Localize Element Vector from a Vect instance.
 - void `ElementNodeVector` (`const Vect< real_t > &b`, `LocalVect< real_t, NEE_ > &be`)
Localize Element Vector from a Vect instance.
 - void `ElementNodeVector` (`const Vect< real_t > &b`, `LocalVect< real_t, NEN_ > &be`, `int dof`)
Localize Element Vector from a Vect instance.
 - void `ElementNodeVectorSingleDOF` (`const Vect< real_t > &b`, `LocalVect< real_t, NEN_ > &be`)
Localize Element Vector from a Vect instance.
 - void `ElementSideVector` (`const Vect< real_t > &b`, `LocalVect< real_t, NSE_ > &be`)
Localize Element Vector from a Vect instance.
 - void `ElementVector` (`const Vect< real_t > &b`, `int dof_type=NODE.FIELD`, `int flag=0`)
Localize Element Vector.
 - void `SideVector` (`const Vect< real_t > &b`)
Localize Side Vector.
 - void `ElementNodeCoordinates` ()
Localize coordinates of element nodes.
 - void `SideNodeCoordinates` ()
Localize coordinates of side nodes.
 - void `ElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one.
 - void `ElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one.
 - void `ElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one.
 - void `ElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one.
 - void `ElementAssembly` (`TrMatrix< real_t > &A`)
Assemble element matrix into global one.
 - void `ElementAssembly` (`Vect< real_t > &v`)
Assemble element vector into global one.
 - void `DGElementAssembly` (`Matrix< real_t > *A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
 - void `DGElementAssembly` (`SkSMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
 - void `DGElementAssembly` (`SkMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
 - void `DGElementAssembly` (`SpMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
 - void `DGElementAssembly` (`TrMatrix< real_t > &A`)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
 - void `SideAssembly` (`Matrix< real_t > *A`)
Assemble side (edge or face) matrix into global one.
 - void `SideAssembly` (`SkSMatrix< real_t > &A`)
Assemble side (edge or face) matrix into global one.
 - void `SideAssembly` (`SkMatrix< real_t > &A`)

- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`SpMatrix< real_t > &A`)
- Assemble side (edge or face) matrix into global one.*
 - void `SideAssembly` (`Vect< real_t > &v`)
- Assemble side (edge or face) vector into global one.*
 - void `AxbAssembly` (const `Element` &el, const `Vect< real_t > &x`, `Vect< real_t > &b`)
- Assemble product of element matrix by element vector into global vector.*
 - void `AxbAssembly` (const `Side` &sd, const `Vect< real_t > &x`, `Vect< real_t > &b`)
- Assemble product of side matrix by side vector into global vector.*
 - size_t `getNbNodes` () const
- Return number of element nodes.*
 - size_t `getNbEq` () const
- Return number of element equations.*
 - `real_t * A` ()
- Return element matrix as a C-array.*
 - `real_t * sA` ()
- Return side matrix as a C-array.*
 - `real_t * b` ()
- Return element right-hand side as a C-array.*
 - `real_t * sb` ()
- Return side right-hand side as a C-array.*
 - `real_t * Prev` ()
- Return element matrix as a C-array.*
 - `LocalMatrix< real_t, NEE_, NEE_ > & EA` ()
- Return element matrix as a LocalMatrix instance.*
 - `LocalMatrix< real_t, NSE_, NSE_ > & SA` ()
- Return side matrix as a LocalMatrix instance.*
 - `LocalVect< real_t, NEE_ > & Eb` ()
- Return element right-hand side as a LocalVect instance.*
 - `LocalVect< real_t, NEE_ > & Ep` ()
- Return element matrix as a C-array.*
 - void `setInitialSolution` (const `Vect< real_t > &u`)
- Set initial solution (previous time step)*
 - `real_t setMaterialProperty` (const string &exp, const string &prop)
- Define a material property by an algebraic expression.*
 - void `setMesh` (class `Mesh` &m)
- Define mesh and renumber DOFs after removing imposed ones.*
 - `Mesh & getMesh` () const
- Return reference to Mesh instance.*
 - `LinearSolver< real_t > & getLinearSolver` ()
- Return reference to linear solver instance.*
 - void `setSolver` (int ls, int pc=IDENT_PREC)
- Choose solver for the linear system.*
 - int `solveEigenProblem` (int nb_eigv, bool g=false)
- Compute eigenvalues and eigenvectors.*
 - `real_t getEigenValue` (int n) const

Return the n -th eigenvalue.

- void `getEigVector` (int n , `Vect`< `real_t` > & v) const

Store the eigenvector corresponding to a given eigenvalue.

- class `Eigen` & `getEigenSolver` ()

Return reference to eigenproblem solver.

Protected Member Functions

- void `Viscosity` (const `real_t` & $visc$)

Set (constant) Viscosity.

- void `Viscosity` (const string & exp)

Set viscosity given by an algebraic expression.

- void `Density` (const `real_t` & $dens$)

Set (constant) Viscosity.

- void `Density` (const string & exp)

Set Density given by an algebraic expression.

- void `ThermalExpansion` (const `real_t` * e)

Set (constant) thermal expansion coefficient.

- void `ThermalExpansion` (const string & exp)

Set thermal expansion coefficient given by an algebraic expression.

- void `setMaterial` ()

Set material properties.

- void `Init` (const `Element` * el)

Set element arrays to zero.

- void `Init` (const `Side` * sd)

Set side arrays to zero.

7.74.1 Detailed Description

Builds finite element arrays for incompressible Navier-Stokes equations in 2-D domains using Q_1/P_0 element and a penalty formulation for the incompressibility condition.

7.74.2 Constructor & Destructor Documentation

`NSP2DQ41` ()

Default Constructor.

Builds an empty equation

`NSP2DQ41` (const `Element` * el)

Constructor using `Element` data.

Parameters

in	el	Pointer to <code>Element</code> instance
------	------	--

`NSP2DQ41` (const `Side` * sd)

Constructor using `Side` data.

Parameters

<i>in</i>	<i>sd</i>	Pointer to Side instance
-----------	-----------	--

NSP2DQ41 (const Element * el, const Vect< real_t > & u, const real_t & time = 0.)

Constructor using element and previous time data.

Parameters

<i>in</i>	<i>el</i>	Pointer to Element instance
<i>in</i>	<i>u</i>	Vector that contains velocity at previous time step
<i>in</i>	<i>time</i>	Time value [Default: 0.]

NSP2DQ41 (const Side * sd, const Vect< real_t > & u, const real_t & time = 0.)

Constructor using side and previous time data.

Parameters

<i>in</i>	<i>sd</i>	Pointer to Side instance
<i>in</i>	<i>u</i>	Vector that contains velocity at previous time step
<i>in</i>	<i>time</i>	Time value [Default: 0.]

7.74.3 Member Function Documentation

void LHS1_Convection (real_t coef = 1.)

Add convection contribution to left-hand side after multiplication by coef [Default: 1].

First term, explicit velocity, implicit velocity derivatives

void LHS2_Convection (real_t coef = 1.)

Add convection contribution to left-hand side after multiplication by coef [Default: 1].

Second term, implicit velocity, explicit velocity derivatives

void BodyRHS (UserData< real_t > & ud)

Add body right-hand side term to right-hand side.

Parameters

<i>in</i>	<i>ud</i>	UserData instance that defines data
-----------	-----------	---

void BoundaryRHS (UserData< real_t > & ud)

Add boundary right-hand side term to right-hand side.

Parameters

<i>in</i>	<i>ud</i>	UserData instance that defines data
-----------	-----------	---

void Periodic (real_t coef = 1.e20)

Add contribution of periodic boundary condition (by a penalty technique).

Boundary nodes where periodic boundary conditions are to be imposed must have codes equal to PERIODIC_A on one side and PERIODIC_B on the opposite side.

Parameters

<i>in</i>	<i>coef</i>	Value of penalty parameter [Default: 1.e20].
-----------	-------------	--

void updateBC (const Element & el, const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>el</i>	Reference to current element instance
<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< real_t > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

<i>in</i>	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

<i>in</i>	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <i>NODE_FIELD</i>, DOFs are supported by nodes [Default] • <i>ELEMENT_FIELD</i>, DOFs are supported by elements • <i>SIDE_FIELD</i>, DOFs are supported by sides
<i>in</i>	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • = 0, All DOFs are taken into account [Default] • != 0, Only DOF No. <i>dof</i> is handled in the system

void LocalNodeVector (Vect< real_t > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <i>ePrev</i> . This member function is to be used if a constructor with <i>Element</i> was invoked.
-----------	----------	---

void ElementNodeVector (const Vect< real_t > & b, LocalVect< real_t , NEE_ > & be)
[inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.
<i>in</i>	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only the degree of freedom is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< real.t > & b, LocalVect< real.t , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector *b* is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< real.t > & b, LocalVect< real.t , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

<i>in</i>	<i>b</i>	Global vector to be localized.
<i>out</i>	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< real.t > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF number <code>dof</code> is handled in the system The resulting local vector can be accessed by attribute <code>ePrev</code> .

Remarks

This member function is to be used if a constructor with `Element` was invoked. It uses the `Element` pointer `_theElement`

void SideVector (const Vect< real.t > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< real_t > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< real_t > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
--	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

	A	Global matrix stored as an SkSMatrix instance
--	---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< real_t > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< real_t > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< real_t > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< real_t > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< real_t > & x, Vect< real_t > & b)
[inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real_t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function `getEigen` was called with an argument `nb_eigv` greater or equal to `n`. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

Parameters

in	<i>n</i>	Label of the eigenvalue
out	<i>v</i>	Vect instance containing the corresponding eigenvector. This vector is resized.

7.75 Partition Class Reference

To partition a finite element mesh into balanced submeshes.

Public Member Functions

- [Partition](#) ([Mesh](#) &mesh, char *file)
- [Partition](#) ([Mesh](#) &mesh, size_t n)
Constructor to partition a mesh into n submeshes.
- [~Partition](#) ()
Destructor.
- size_t [getNbSubMeshes](#) () const
Return number of submeshes.
- size_t [getNbNodes](#) (size_t i) const
Return number of nodes in given submesh.
- size_t [getNbElements](#) (size_t i) const
Return number of elements in given submesh.
- [Mesh](#) & [getMesh](#) (size_t i) const
Return the submesh of label i.
- size_t [getNodeLabelInSubMesh](#) (size_t sm, size_t label) const
Return node label in subdomain by giving its label in initial mesh.
- size_t [getElementLabelInSubMesh](#) (size_t sm, size_t label) const
Return element label in subdomain by giving its label in initial mesh.
- size_t [getNodeLabelInMesh](#) (size_t sm, size_t label) const
Return node label in initial mesh by giving its label in submesh.
- size_t [getElementLabelInMesh](#) (size_t sm, size_t label) const
Return element label in initial mesh by giving its label in submesh.
- size_t [getNbInterfaceSides](#) (size_t sm) const
Return Nb. of interface sides for a given sub-mesh.
- size_t [getFirstSideLabel](#) (size_t sm, size_t i) const
Return i-th side label in sub-mesh sm
- size_t [getSubMesh](#) (size_t sm, size_t i) const
Return submesh that contains the i-th side label in sub-mesh sm
- size_t [getSecondSideLabel](#) (size_t sm, size_t i) const
- void [setVerbose](#) (size_t verb)
Set Message Level.

- void `Put` (size_t n, char *file) const
Save n-th submesh in file.
- void `Init` (int *epart)
Get interface information.

Friends

- ostream & `operator<<` (ostream &s, const `Partition` &p)
Output class information.

7.75.1 Detailed Description

To partition a finite element mesh into balanced submeshes.

Class `Partition` enables partitioning a given mesh into a given number of submeshes with a minimal connectivity. `Partition` uses the well known `metis` library that is included in the `OFELI` library. A more detailed description of `metis` can be found in the web site:

http://www.csit.fsu.edu/~burkardt/c_src/metis/metis.html

7.75.2 Constructor & Destructor Documentation

`Partition` (`Mesh` & mesh, char * file)

Constructor to get interface information. `Partition` information is read in file

7.75.3 Member Function Documentation

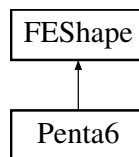
size_t `getSecondSideLabel` (size_t sm, size_t i) const

Return side label in the neighbouring submesh corresponding to i-th side label in sub-mesh sm

7.76 Penta6 Class Reference

Defines a 6-node pentahedral finite element using P_1 interpolation in local coordinates (s.x,s.y) and Q_1 isoparametric interpolation in local coordinates (s.x,s.z) and (s.y,s.z).

Inheritance diagram for `Penta6`:



Public Member Functions

- `Penta6` ()
Default Constructor.
- `Penta6` (const `Element` *element)
Constructor when data of `Element` el are given.
- `~Penta6` ()
Destructor.
- void `set` (const `Element` *el)

- Choose element by giving its pointer.*

 - void `setLocal` (const `Point< real_t >` &s)

Initialize local point coordinates in element.
- `Point< real_t > DSh` (size_t i) const

Return derivatives of shape function of node i at a given point.
- `real_t getMaxEdgeLength` () const

Return Maximum length of pentahedron edges.
- `real_t getMinEdgeLength` () const

Return Mimimum length of pentahedron edges.
- `real_t Sh` (size_t i) const

Return shape function of node i at given point.
- `real_t Sh` (size_t i, `Point< real_t >` s) const

Calculate shape function of node i at a given point s.
- `real_t getDet` () const

Return determinant of jacobian.
- `Point< real_t > getCenter` () const

Return coordinates of center of element.
- `Point< real_t > getLocalPoint` () const

Localize a point in the element.
- `Point< real_t > getLocalPoint` (const `Point< real_t >` &s) const

Localize a point in the element.

7.76.1 Detailed Description

Defines a 6-node pentahedral finite element using P_1 interpolation in local coordinates (s.x,s.y) and Q_1 isoparametric interpolation in local coordinates (s.x,s.z) and (s.y,s.z).

The reference element is the cartesian product of the standard reference triangle with the line $[-1,1]$. The nodes are ordered as follows: **Node 1** in reference element is at s=(1,0,0) **Node 2** in reference element is at s=(0,1,0) **Node 3** in reference element is at s=(0,0,0) **Node 4** in reference element is at s=(1,0,1) **Node 5** in reference element is at s=(0,1,1) **Node 6** in reference element is at s=(0,0,1)

The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function `setLocal()` must be invoked.

7.76.2 Constructor & Destructor Documentation

Penta6 (const `Element` * element)

Constructor when data of `Element` e1 are given.

Parameters

in	element	Pointer to <code>Element</code>
----	---------	---------------------------------

7.76.3 Member Function Documentation

void setLocal (const `Point< real_t >` & s)

Initialize local point coordinates in element.

Parameters

in	s	Point in the reference element This function computes jacobian, shape functions and their partial derivatives at s. Other member functions only return these values.
----	---	--

Point<real_t> DSh (size_t i) const

Return derivatives of shape function of node i at a given point.

Member function [setLocal\(\)](#) must have been called before in order to calculate relevant quantities.

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

in	i	Local node label
in	s	Point in the reference triangle where the shape function is evaluated

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > &s) const [inherited]

Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.77 PhaseChange Class Reference

This class enables defining phase change laws for a given material.

Public Member Functions

- virtual [~PhaseChange](#) ()
Destructor.
- int [E2T](#) (real_t &H, real_t &T, real_t &gamma)
Calculate temperature from enthalpy.
- virtual int [EnthalpyToTemperature](#) (real_t &H, real_t &T, real_t &gamma)
Virtual function to calculate temperature from enthalpy.
- void [setMaterial](#) ([Material](#) &m, int code)
Choose [Material](#) instance and material code.

- [Material](#) & [getMaterial](#) () const

Return reference to [Material](#) instance.

7.77.1 Detailed Description

This class enables defining phase change laws for a given material.

These laws are predefined for a certain number of materials. The user can set himself a specific behavior for his own materials by defining a class that inherits from [PhaseChange](#). The derived class must have at least the member function

```
int EnthalpyToTemperature(real_t &H, real_t &T, real_t &gamma)
```

7.77.2 Member Function Documentation

```
int E2T ( real_t & H, real_t & T, real_t & gamma )
```

Calculate temperature from enthalpy.

This member function is to be called in any equation class that needs phase change laws.

Parameters

in	H	Enthalpy value
out	T	Calculated temperature value
out	γ	Maximal slope of the curve $H \rightarrow T$

```
virtual int EnthalpyToTemperature ( real_t & H, real_t & T, real_t & gamma ) [virtual]
```

Virtual function to calculate temperature from enthalpy.

This member function must be implemented in any derived class in order to define user's own material laws.

Parameters

in	H	Enthalpy value
out	T	Calculated temperature value
out	γ	Maximal slope of the curve $H \rightarrow T$

7.78 Point< T_ > Class Template Reference

Defines a point with arbitrary type coordinates.

Public Member Functions

- [Point](#) ()

Default constructor.

- [Point](#) (T_ a, T_ b=T_(0), T_ c=T_(0))

Constructor that assigns a, b to x-, y- and z-coordinates respectively.

- [Point](#) (const [Point](#)< T_ > &p)

Copy constructor.

- T_ & [operator](#)() (size_t i)

Operator (): Non constant version.

- const T_ & [operator](#)() (size_t i) const

Operator (): Constant version.

- T_ & [operator](#)[] (size_t i)

- Operator []: Non constant version.*
 - `const T_ & operator[] (size_t i) const`
 - Operator []: Constant version.*
 - `Point< T_ > & operator+= (const Point< T_ > &p)`
 - Operator +=*
 - `Point< T_ > & operator-= (const Point< T_ > &p)`
 - Operator -=*
 - `Point< T_ > & operator= (const T_ &a)`
 - Operator =*
 - `Point< T_ > & operator+= (const T_ &a)`
 - Operator +=*
 - `Point< T_ > & operator-= (const T_ &a)`
 - Operator -=*
 - `Point< T_ > & operator*= (const T_ &a)`
 - Operator *=*
 - `Point< T_ > & operator/= (const T_ &a)`
 - Operator /=*
 - `bool operator== (const Point< T_ > &p)`
 - Operator ==*
 - `bool operator!= (const Point< T_ > &p)`
 - Operator !=*
 - `double NNorm () const`
 - Return squared euclidean norm of vector.*
 - `double Norm () const`
 - Return norm (length) of vector.*
 - `void Normalize ()`
 - Normalize vector.*
 - `Point< double > Director (const Point< double > &p) const`
 - Return Director (Normalized vector)*
 - `bool isCloseTo (const Point< double > &a, double toler=OFELI.TOLERANCE) const`
 - Return true if current point is close to instance a (up to tolerance toler)*
 - `T_ operator, (const Point< T_ > &p) const`
 - Return Dot (scalar) product of two vectors.*

Public Attributes

- `T_ x`
First coordinate.
- `T_ y`
Second coordinate.
- `T_ z`
Third coordinate.

7.78.1 Detailed Description

`template<class T_>class OFELI::Point< T_ >`

Defines a point with arbitrary type coordinates.
Operators = and () are overloaded.

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
-------------------------	---

7.78.2 Constructor & Destructor Documentation

Point (T_ a, T_ b = T_(0), T_ c = T_(0))

Constructor that assigns a, b to x-, y- and z-coordinates respectively.
Default values for b and c are 0

7.78.3 Member Function Documentation

T_& operator() (size_t i)

Operator (): Non constant version.
Values i = 1, 2, 3 correspond to x, y and z respectively

const T_& operator() (size_t i) const

Operator (): Constant version.
Values i = 1, 2, 3 correspond to x, y and z respectively

T_& operator[] (size_t i)

Operator []: Non constant version.
Values i = 0, 1, 2 correspond to x, y and z respectively

const T_& operator[] (size_t i) const

Operator []: Constant version.
Values i = 0, 1, 2 correspond to x, y and z respectively

Point<T_>& operator+=(const Point< T_ > & p)

Operator +=
Add point p to current instance

Point<T_>& operator-= (const Point< T_ > & p)

Operator -=
Subtract point p from current instance

Point<T_>& operator= (const T_ & a)

Operator =
Assign constant a to current instance coordinates

Point<T_>& operator+=(const T_ & a)

Operator +=
Add constant a to current instance coordinates

Point<T_>& operator-= (const T_ & a)

Operator -=
Subtract constant a from current instance coordinates

Point<T_>& operator*= (const T_ & a)

Operator *=
Multiply constant a by current instance coordinates

Point<T_>& operator/= (const T_ & a)

Operator /=
Divide current instance coordinates by a

bool operator== (const Point< T_ > & p)

Operator ==
Return true if current instance is equal to p, false otherwise.

bool operator!= (const Point< T_ > & p)

Operator !=
Return false if current instance is equal to p, true otherwise.

void Normalize ()

Normalize vector.
Divide vector components by its 2-norm

bool isCloseTo (const Point< double > & a, double toler = OFELI_TOLERANCE) const

Return true if current point is close to instance a (up to tolerance toler)
Default value for toler is the OFELI_TOLERANCE constant.

T_ operator, (const Point< T_ > & p) const

Return Dot (scalar) product of two vectors.
A typical use of this operator is `double a = (p,q)` where p and q are 2 instances of `Point<double>`
Parameters

in	p	Point instance by which the current instance is multiplied
----	---	--

7.79 Point2D< T_ > Class Template Reference

Defines a 2-D point with arbitrary type coordinates.

Public Member Functions

- [Point2D](#) ()
Default constructor.
- [Point2D](#) (T_ a, T_ b=T_(0))
Constructor that assigns a, b to x-, y- and y-coordinates respectively.
- [Point2D](#) (T_ *a)
Initialize point coordinates with C-array a.
- [Point2D](#) (const [Point2D](#)< T_ > &pt)
Copy constructor.
- [Point2D](#) (const [Point](#)< T_ > &pt)

Copy constructor from class [Point](#).

- `T_ & operator() (size_t i)`
Operator(): Non constant version.
- `const T_ & operator() (size_t i) const`
Operator(): Constant version.
- `T_ & operator[] (size_t i)`
Operator[]: Non constant version.
- `const T_ & operator[] (size_t i) const`
Operator[] Constant version.
- `Point2D< T_ > & operator= (const Point2D< T_ > &p)`
Operator =
- `Point2D< T_ > & operator+= (const Point2D< T_ > &p)`
Operator +=
- `Point2D< T_ > & operator-= (const Point2D< T_ > &p)`
Operator -=
- `Point2D< T_ > & operator= (const T_ &a)`
Operator =
- `Point2D< T_ > & operator+= (const T_ &a)`
Operator +=
- `Point2D< T_ > & operator-= (const T_ &a)`
Operator -=
- `Point2D< T_ > & operator*= (const T_ &a)`
*Operator *=*
- `Point2D< T_ > & operator/= (const T_ &a)`
Operator /=
- `bool operator== (const Point2D< T_ > &p)`
Operator ==
- `bool operator!= (const Point2D< T_ > &p)`
Operator !=
- `real_t CrossProduct (const Point2D< real_t > &lp, const Point2D< real_t > &rp)`
Return Cross product of two vectors lp and rp
- `real_t NNorm () const`
Return squared norm (length) of vector.
- `real_t Norm () const`
Return norm (length) of vector.
- `Point2D< real_t > Director (const Point2D< real_t > &p) const`
Return Director (Normalized vector)
- `bool isCloseTo (const Point2D< real_t > &a, real_t toler=OFELI_TOLERANCE) const`
Return true if current point is close to instance a (up to tolerance toler)

Public Attributes

- `T_ x`
First coordinate of point.
- `T_ y`
Second coordinate of point.

7.79.1 Detailed Description

template<class T_>class OFELI::Point2D< T_ >

Defines a 2-D point with arbitrary type coordinates.

Operators = and () are overloaded. The actual
Template Parameters

< T_ >	Data type (real_t, float, complex<real_t>, ...)
---------------------	---

7.79.2 Constructor & Destructor Documentation

Point2D (T_ a, T_ b = T_(0))

Constructor that assigns a, b to x-, y- and y-coordinates respectively.

Default value for *b* is 0

7.79.3 Member Function Documentation

T_& operator() (size_t i)

Operator() : Non constant version.

Values *i* = 1,2 correspond to x and y respectively

const T_& operator() (size_t i) const

Operator() : Constant version.

Values *i*=1,2 correspond to x and y respectively

T_& operator[] (size_t i)

Operator[]: Non constant version.

Values *i*=0,1 correspond to x and y respectively

const T_& operator[] (size_t i) const

Operator[] Constant version.

Values *i*=0,1 correspond to x and y respectively

Point2D<T_>& operator= (const Point2D< T_ > & p)

Operator =

Assign point *p* to current instance

Point2D<T_>& operator+= (const Point2D< T_ > & p)

Operator +=

Add point *p* to current instance

Point2D<T_>& operator-= (const Point2D< T_ > & p)

Operator -=

Subtract point *p* from current instance

Point2D<T_>& operator= (const T_ & a)

Operator =

Assign constant *a* to current instance coordinates

Point2D<T_>& operator+= (const T_ & a)

Operator +=

Add constant a to current instance coordinates

Point2D<T_>& operator-= (const T_ & a)

Operator -=

Subtract constant a from current instance coordinates

Point2D<T_>& operator*= (const T_ & a)

Operator *=

Multiply constant a by current instance coordinates

Point2D<T_>& operator/= (const T_ & a)

Operator /=

Divide current instance coordinates by a

bool operator== (const Point2D< T_ > & p)

Operator ==

Return true if current instance is equal to p, false otherwise.

bool operator!= (const Point2D< T_ > & p)

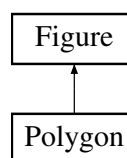
Operator !=

Return false if current instance is equal to p, true otherwise.

7.80 Polygon Class Reference

To store and treat a polygonal figure.

Inheritance diagram for Polygon:



Public Member Functions

- `polygon ()`
Default constructor.
- `Polygon (const Vect< Point< double > > &v, int code=1)`
Constructor.
- `void setVertices (const Vect< Point< double > > &v)`
Assign vertices of polygon.
- `double getSignedDistance (const Point< double > &p) const`
Return signed distance of a given point from the current polygon.
- `Polygon & operator+= (Point< double > a)`

Operator +=.

- **Polygon** & **operator+=** (double a)

*Operator *.=.*

- void **setCode** (int code)

Choose a code for the domain defined by the figure.

- void **getSignedDistance** (const **Grid** &g, **Vect**< double > &d) const

Calculate signed distance to current figure with respect to grid points.

- double **dLine** (const **Point**< double > &p, const **Point**< double > &a, const **Point**< double > &b) const

Compute signed distance from a line.

7.80.1 Detailed Description

To store and treat a polygonal figure.

7.80.2 Constructor & Destructor Documentation

Polygon (const **Vect**< **Point**< double > > &v, int code = 1)

Constructor.

Parameters

in	v	Vect instance containing list of coordinates of polygon vertices
in	code	Code to assign to the generated domain (Default value = 1)

7.80.3 Member Function Documentation

void setVertices (const **Vect**< **Point**< double > > &v)

Assign vertices of polygon.

Parameters

in	v	Vector containing vertices coordinates in counter clockwise order
----	---	---

double getSignedDistance (const **Point**< double > &p) const [virtual]

Return signed distance of a given point from the current polygon.

The computed distance is negative if p lies in the polygon, negative if it is outside, and 0 on its boundary

Parameters

in	p	Point <double> instance
----	---	--------------------------------

Reimplemented from [Figure](#).

Polygon& operator+= (**Point**< double > a)

Operator +=.

Translate polygon by a vector a

Polygon& operator+= (double a)

Operator *.=.

Scale polygon by a factor a

void getSignedDistance (const Grid & g, Vect< double > & d) const [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	g	Grid instance
in	d	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector d doesn't need to be sized before invoking this function

double dLine (const [Point](#)< double > & p, const [Point](#)< double > & a, const [Point](#)< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	p	Point for which distance is computed
in	a	First vertex of line
in	b	Second vertex of line

Returns

Signed distance

7.81 $\text{Prec}<T_>$ Class Template Reference

To set a preconditioner.

Public Member Functions

- [Prec](#) ()
Default constructor.
- [Prec](#) (int type)
Constructor that chooses preconditioner.
- [Prec](#) (const [SpMatrix](#)< $T_$ > & A , int type=[IDENT_PREC](#))
Constructor using matrix of the linear system to precondition.
- [Prec](#) (const [Matrix](#)< $T_$ > * A , int type=[IDENT_PREC](#))
Constructor using matrix of the linear system to precondition.
- [~Prec](#) ()
Destructor.
- void [setType](#) (int type)
Define preconditioner type.
- void [setMatrix](#) (const [Matrix](#)< $T_$ > * A)
Define pointer to matrix for preconditioning (if this one is abstract)
- void [setMatrix](#) (const [SpMatrix](#)< $T_$ > & A)
Define the matrix for preconditioning.
- void [Solve](#) ([Vect](#)< $T_$ > & x) const
Solve a linear system with preconditioning matrix.
- void [Solve](#) (const [Vect](#)< $T_$ > & b , [Vect](#)< $T_$ > & x) const
Solve a linear system with preconditioning matrix.
- void [TransSolve](#) ([Vect](#)< $T_$ > & x) const

Solve a linear system with transposed preconditioning matrix.

- void **TransSolve** (const **Vect**< T_ > &b, **Vect**< T_ > &x) const

Solve a linear system with transposed preconditioning matrix.

- T_ & **getPivot** (size_t i) const

Return i-th pivot of preconditioning matrix.

7.81.1 Detailed Description

template<class T_>class OFELI::Prec< T_ >

To set a preconditioner.

The preconditioner type is chosen in the constructor

Template Parameters

< T_ >	Data type (real_t, float, complex<real_t>, ...)
--------	---

7.81.2 Constructor & Destructor Documentation

Prec (int type)

Constructor that chooses preconditioner.

Parameters

in	type	Preconditioner type:
		<ul style="list-style-type: none"> • IDENT_PREC: Identity preconditioner (No preconditioning) • DIAG_PREC: Diagonal preconditioner • ILU_PREC: Icomplete factorization preconditioner • SSOR_PREC: SSOR preconditioner

Prec (const SpMatrix< T_ > & A, int type = IDENT_PREC)

Constructor using matrix of the linear system to precondition.

Parameters

in	A	Matrix to precondition
in	type	Preconditioner type:
		<ul style="list-style-type: none"> • IDENT_PREC: Identity preconditioner (No preconditioning) • DIAG_PREC: Diagonal preconditioner • ILU_PREC: Icomplete factorization preconditioner • SSOR_PREC: SSOR preconditioner

Prec (const Matrix< T_ > * A, int type = IDENT_PREC)

Constructor using matrix of the linear system to precondition.

Parameters

in	<i>A</i>	Pointer to abstract Matrix class to precondition
in	<i>type</i>	Preconditioner type: <ul style="list-style-type: none"> • IDENT_PREC: Identity preconditioner (No preconditioning) • DIAG_PREC: Diagonal preconditioner • ILU_PREC: lcomplete factorization preconditioner • SSOR_PREC: SSOR preconditioner

7.81.3 Member Function Documentation

void setType (int type)

Define preconditioner type.

Parameters

in	<i>type</i>	Preconditioner type: <ul style="list-style-type: none"> • IDENT_PREC: Identity preconditioner (No preconditioning) • DIAG_PREC: Diagonal preconditioner • ILU_PREC: lcomplete factorization preconditioner • SSOR_PREC: SSOR preconditioner
-----------	-------------	---

void setMatrix (const Matrix< $T_$ > * A)

Define pointer to matrix for preconditioning (if this one is abstract)

Parameters

in	<i>A</i>	Matrix to precondition
-----------	----------	------------------------

void setMatrix (const SpMatrix< $T_$ > & A)

Define the matrix for preconditioning.

Parameters

in	<i>A</i>	Matrix to precondition (instance of class SpMatrix)
-----------	----------	--

void Solve (Vect< $T_$ > & x) const

Solve a linear system with preconditioning matrix.

Parameters

in,out	<i>x</i>	Right-hand side in input and solution in output.
---------------	----------	--

void Solve (const Vect< $T_$ > & b, Vect< $T_$ > & x) const

Solve a linear system with preconditioning matrix.

Parameters

in	b	Right-hand side
out	x	Solution vector

void TransSolve (Vect< T_ > & x) const

Solve a linear system with transposed preconditioning matrix.

Parameters

in,out	x	Right-hand side in input and solution in output.
--------	-----	--

void TransSolve (const Vect< T_ > & b, Vect< T_ > & x) const

Solve a linear system with transposed preconditioning matrix.

Parameters

in	b	Right-hand side vector
out	x	Solution vector

7.82 Precond< T_ > Class Template Reference

An abstract class from which derive all preconditioning classes.

Public Member Functions

- [Precond](#) ()
Default constructor.
- [Precond](#) (const [Matrix](#)< T_ > *A)
Constructor using matrix (A pointer to abstract [Matrix](#) class)
- virtual [~Precond](#) ()
Destructor.

7.82.1 Detailed Description

template<class T_>class OFELI::Precond< T_ >

An abstract class from which derive all preconditioning classes.

Template Parameters

< T_ >	Data type (real_t, float, complex<real_t>, ...)
--------	---

7.83 Prescription Class Reference

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

Public Member Functions

- [Prescription](#) ()
Default constructor.
- [Prescription](#) ([Mesh](#) &mesh, const string &file)
Constructor that gives an instance of class [Mesh](#) and the data file name.
- [~Prescription](#) ()
Destructor.
- int [get](#) (int type, [Vect](#)< [real_t](#) > &v, [real_t](#) time=0, size_t dof=0)

7.83.1 Detailed Description

To prescribe various types of data by an algebraic expression. Data may consist in boundary conditions, forces, tractions, fluxes, initial condition. All these data types can be defined through an enumerated variable.

7.83.2 Constructor & Destructor Documentation

[Prescription](#) ([Mesh](#) & mesh, const string & file)

Constructor that gives an instance of class [Mesh](#) and the data file name.

It reads parameters in [Prescription](#) Format from this file.

Parameters

in	<i>mesh</i>	Mesh instance
in	<i>file</i>	Name of Prescription file

7.83.3 Member Function Documentation

int [get](#) (int type, [Vect](#)< [real_t](#) > & v, [real_t](#) time = 0, size_t dof = 0)

Read data in the given file and stores in a [Vect](#) instance for a chosen DOF. The input value type determines the type of data to read.

Parameters

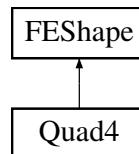
in	<i>type</i>	Type of data to seek. To choose among the enumerated values: <ul style="list-style-type: none"> • BOUNDARY_CONDITION: Read values for (Dirichlet) boundary conditions • BOUNDARY_FORCE: Read values for boundary force (Neumann boundary condition). The values TRACTION and FLUX have the same effect. • BODY_FORCE: Read values for body (or volume) forces. The value SOURCE has the same effect. • POINT_FORCE: Read values for pointwise forces • INITIAL_FIELD: Read values for initial solution
----	-------------	---

in,out	v	Vect instance that is instantiatd on input and filled on output
in	time	Value of time for which data is read [Default: 0].
in	dof	DOF to store (Default is 0: All DOFs are chosen).

7.84 Quad4 Class Reference

Defines a 4-node quadrilateral finite element using Q_1 isoparametric interpolation.

Inheritance diagram for Quad4:



Public Member Functions

- [Quad4](#) ()
Default Constructor.
- [Quad4](#) (const [Element](#) *element)
Constructor when data of [Element](#) el are given.
- [Quad4](#) (const [Side](#) *side)
Constructor when data of [Side](#) sd are given.
- [~Quad4](#) ()
Destructor.
- void [set](#) (const [Element](#) *el)
Choose element by giving its pointer.
- void [set](#) (const [Side](#) *sd)
Choose side by giving its pointer.
- void [setLocal](#) (const [Point](#)< [real_t](#) > &s)
Initialize local point coordinates in element.
- [Point](#)< [real_t](#) > [DSh](#) (size_t i) const
Return derivatives of shape function of node i at a given point.
- [Point](#)< [real_t](#) > [Grad](#) (const [LocalVect](#)< [real_t](#), 4 > &u, const [Point](#)< [real_t](#) > &s)
Return gradient of a function defined at element nodes.
- [real_t](#) [getMaxEdgeLength](#) () const
Return maximal edge length of quadrilateral.
- [real_t](#) [getMinEdgeLength](#) () const
Return minimal edge length of quadrilateral.
- [real_t](#) [Sh](#) (size_t i) const
Return shape function of node i at given point.
- [real_t](#) [Sh](#) (size_t i, [Point](#)< [real_t](#) > s) const
Calculate shape function of node i at a given point s.
- [real_t](#) [getDet](#) () const
Return determinant of jacobian.
- [Point](#)< [real_t](#) > [getCenter](#) () const
Return coordinates of center of element.

- `Point< real_t > getLocalPoint ()` const
Localize a point in the element.
- `Point< real_t > getLocalPoint (const Point< real_t > &s)` const
Localize a point in the element.

7.84.1 Detailed Description

Defines a 4-node quadrilateral finite element using Q_1 isoparametric interpolation.

The reference element is the square $[-1,1] \times [-1,1]$. The user must take care to the fact that determinant of jacobian and other quantities depend on the point in the reference element where they are calculated. For this, before any utilization of shape functions or jacobian, function `setLocal()` must be invoked.

7.84.2 Constructor & Destructor Documentation

Quad4 (const Element * element)

Constructor when data of `Element` e1 are given.

Parameters

<code>in</code>	<code>element</code>	Pointer to <code>Element</code>
-----------------	----------------------	---------------------------------

Quad4 (const Side * side)

Constructor when data of `Side` sd are given.

Parameters

<code>in</code>	<code>side</code>	Pointer to <code>Side</code>
-----------------	-------------------	------------------------------

7.84.3 Member Function Documentation

void setLocal (const Point< real_t > & s)

Initialize local point coordinates in element.

Parameters

<code>in</code>	<code>s</code>	<code>Point</code> in the reference element This function computes jacobian, shape functions and their partial derivatives at s. Other member functions only return these values.
-----------------	----------------	---

Point<real_t> DSh (size_t i) const

Return derivatives of shape function of node i at a given point.

Member function `setLocal()` must have been called before in order to calculate relevant quantities.

Point<real_t> Grad (const LocalVect< real_t, 4 > & u, const Point< real_t > & s)

Return gradient of a function defined at element nodes.

Parameters

in	u	Vector of values at nodes
in	s	Local coordinates (in $[-1, 1]$) of point where the gradient is evaluated

Returns

Value of gradient

Note

If the derivatives of shape functions were not computed before calling this function (by calling `setLocal`), this function will compute them

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s .

Parameters

in	i	Local node label
in	s	Point in the reference triangle where the shape function is evaluated

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element \rightarrow Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element \rightarrow Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.85 Reconstruction Class Reference

To perform various reconstruction operations.

Public Member Functions

- [Reconstruction](#) ()
Default constructor.
- [Reconstruction](#) (const [Mesh](#) &ms)
Constructor using a reference to a [Mesh](#) instance.
- [~Reconstruction](#) ()
Destructor.
- void [setMesh](#) ([Mesh](#) &ms)
Provide [Mesh](#) instance.

- void **P0toP1** (const **Vect**< **real_t** > &u, **Vect**< **real_t** > &v)
Smooth an elementwise field to obtain a nodewise field by L^2 projection.
- void **DP1toP1** (const **Vect**< **real_t** > &u, **Vect**< **real_t** > &v)
Smooth an Discontinuous P1 field to obtain a nodewise (Continuous P_1) field by L^2 projection.

7.85.1 Detailed Description

To perform various reconstruction operations.

This class enables various reconstruction operations like smoothing, projections, ...

7.85.2 Member Function Documentation

void P0toP1 (const **Vect< **real_t** > & u, **Vect**< **real_t** > & v)**

Smooth an elementwise field to obtain a nodewise field by L^2 projection.

Parameters

in	<i>u</i>	Vect instance that contains field to smooth
out	<i>v</i>	Vect instance that contains on output smoothed field

void DP1toP1 (const **Vect< **real_t** > & u, **Vect**< **real_t** > & v)**

Smooth an Discontinuous P1 field to obtain a nodewise (Continuous P_1) field by L^2 projection.

Parameters

in	<i>u</i>	Vect instance that contains field to smooth
out	<i>v</i>	Vect instance that contains on output smoothed field

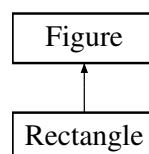
Warning

This function is valid for P_1 triangles (2-D) only.

7.86 Rectangle Class Reference

To store and treat a rectangular figure.

Inheritance diagram for Rectangle:



Public Member Functions

- **Rectangle** ()
Default constructor.
- **Rectangle** (const **Point**< double > &bbm, const **Point**< double > &bbM, int code=1)
Constructor.
- void **setBoundingBox** (const **Point**< double > &bbm, const **Point**< double > &bbM)
Assign bounding box of the rectangle.

- `Point< double > getBoundingBox1 () const`
Return first point of bounding box.
- `Point< double > getBoundingBox2 () const`
Return second point of bounding box.
- `double getSIGNEDDistance (const Point< double > &p) const`
Return signed distance of a given point from the current rectangle.
- `Rectangle & operator+= (Point< double > a)`
Operator +=.
- `Rectangle & operator+= (double a)`
*Operator *+=.*
- `void setCode (int code)`
Choose a code for the domain defined by the figure.
- `void getSIGNEDDistance (const Grid &g, Vect< double > &d) const`
Calculate signed distance to current figure with respect to grid points.
- `double dLine (const Point< double > &p, const Point< double > &a, const Point< double > &b) const`
Compute signed distance from a line.

7.86.1 Detailed Description

To store and treat a rectangular figure.

7.86.2 Constructor & Destructor Documentation

Rectangle (const Point< double > &bbm, const Point< double > &bbM, int code = 1)

Constructor.

Parameters

in	<i>bbm</i>	Left Bottom point of rectangle
in	<i>bbM</i>	Right Top point of rectangle
in	<i>code</i>	Code to assign to rectangle

7.86.3 Member Function Documentation

void setBoundingBox (const Point< double > &bbm, const Point< double > &bbM)

Assign bounding box of the rectangle.

Parameters

in	<i>bbm</i>	Left Bottom point
in	<i>bbM</i>	Right Top point

double getSIGNEDDistance (const Point< double > &p) const [virtual]

Return signed distance of a given point from the current rectangle.

The computed distance is negative if p lies in the rectangle, negative if it is outside, and 0 on its boundary

Parameters

in	<i>p</i>	Point<double> instance
----	----------	------------------------

Reimplemented from [Figure](#).**Rectangle& operator+= (Point< double > a)**

Operator +=.

Translate rectangle by a vector a

Rectangle& operator+= (double a)

Operator *+=.

Scale rectangle by a factor a

void getSignedDistance (const Grid & g, Vect< double > & d) const [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector d doesn't need to be sized before invoking this function

double dLine (const Point< double > & p, const Point< double > & a, const Point< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	<i>p</i>	Point for which distance is computed
in	<i>a</i>	First vertex of line
in	<i>b</i>	Second vertex of line

Returns

Signed distance

7.87 Side Class Reference

To store and treat finite element sides (edges in 2-D or faces in 3-D)

Public Types

- enum [SideType](#) {
[INTERNAL_SIDE](#) = 0,
[EXTERNAL_BOUNDARY](#) = 1,
[INTERNAL_BOUNDARY](#) = 2 }

Public Member Functions

- `Side ()`
Default Constructor.
- `Side (size_t label, const string &shape)`
Constructor initializing side label and shape.
- `Side (size_t label, int shape)`
Constructor initializing side label and shape.
- `Side (const Side &sd)`
Copy constructor.
- `~Side ()`
Destructor.
- `void Add (Node *node)`
Insert a node at end of list of nodes of side.
- `void Add (Edge *edge)`
Insert an edge at end of list of edges of side.
- `void setLabel (size_t i)`
Define label of side.
- `void setFirstDOF (size_t n)`
Define First DOF.
- `void setNbDOF (size_t nb_dof)`
Set number of degrees of freedom (DOF).
- `void DOF (size_t i, size_t dof)`
Define label of DOF.
- `void setDOF (size_t &first_dof, size_t nb_dof)`
Define number of DOF.
- `void setCode (size_t dof, int code)`
Assign code to a DOF.
- `void setCode (const string &exp, int code, size_t dof=1)`
Define code by a boolean algebraic expression invoking coordinates of side nodes.
- `void Replace (size_t label, Node *node)`
Replace a node at a given local label.
- `void Add (Element *el)`
Set pointer to neighbor element.
- `void set (Element *el, size_t i)`
Set pointer to neighbor element.
- `void setNode (size_t i, Node *node)`
Assign a node given by its pointer as the i-th node of side.
- `int getShape () const`
Return side's shape.
- `size_t getLabel () const`
Return label of side.
- `size_t n () const`
Return label of side.
- `size_t getNbNodes () const`
Return number of side nodes.
- `size_t getNbVertices () const`

- Return number of side vertices.*

 - size_t `getNbEq ()` const
- Return number of side equations.*

 - size_t `getNbDOF ()` const
- Return number of DOF.*

 - int `getCode (size_t dof=1)` const
- Return code for a given DOF of node.*

 - size_t `getDOF (size_t i)` const
- Return label of i-th dof.*

 - size_t `getFirstDOF ()` const
- Return label of first dof of node.*

 - `Node *` `getPtrNode (size_t i)` const
- Return pointer to node of local label i.*

 - `Node *` `operator()` (size_t i) const
- Operator ().*

 - size_t `getNodeLabel (size_t i)` const
- Return global label of node with given local label.*

 - `Element *` `getNeighborElement (size_t i)` const
- Return pointer to i-th side neighboring element.*

 - `Element *` `getOtherNeighborElement (Element *el)` const
- Return pointer to other neighboring element than given one.*

 - `Point< real_t >` `getNormal ()` const
- Return normal vector to side.*

 - `Point< real_t >` `getUnitNormal ()` const
- Return unit normal vector to side.*

 - int `isOnBoundary ()` const
- Boundary side or not.*

 - void `setOnBoundary ()`
- Say that the side is on the boundary.*

 - `real_t` `getMeasure ()` const
- Return measure of side.*

 - size_t `Contains (const Node *nd)` const
- Say if a given node belongs to current side.*

 - void `setActive (bool opt=true)`
- Set side is active (default) or not if argument is false*

 - bool `isActive ()` const
- Return true or false whether side is active or not.*

 - int `getLevel ()` const
- Return side level Side level increases when side is refined (starting from 0). If the level is 0, then the element has no father.*

 - void `setChild (Side *sd)`
- Assign side as child of current one and assign current side as father This function is principally used when refining is invoked (e.g. for mesh adaption)*

 - `Side *` `getParent ()` const
- Return pointer to parent side Return null if no parent.*

 - `Side *` `getChild (size_t i)` const
- Return pointer to i-th child side Returns null pointer is no childs.*

 - size_t `getNbChilds ()` const
- Return number of children of side.*

7.87.1 Detailed Description

To store and treat finite element sides (edges in 2-D or faces in 3-D)

Defines a side of a finite element mesh. The sides are given in particular by their shapes and a list of nodes. Each node can be accessed by the member function [getPtrNode\(\)](#). The string defining the element shape must be chosen according to the following list :

7.87.2 Member Enumeration Documentation

enum SideType

To select side type (boundary side or not).

Enumerator

INTERNAL_SIDE Internal side
EXTERNAL_BOUNDARY [Side](#) on external boundary
INTERNAL_BOUNDARY [Side](#) on internal boundary

7.87.3 Constructor & Destructor Documentation

Side (size_t label, const string & shape)

Constructor initializing side label and shape.

Parameters

in	<i>label</i>	Label to assign to side.
in	<i>shape</i>	Shape of side (See class description).

Side (size_t label, int shape)

Constructor initializing side label and shape.

Parameters

in	<i>label</i>	to assign to side.
in	<i>shape</i>	of side (See enum ElementShape in Mesh).

7.87.4 Member Function Documentation

void DOF (size_t i, size_t dof)

Define label of DOF.

Parameters

in	<i>i</i>	DOF index
in	<i>dof</i>	Its label

void setDOF (size_t & first_dof, size_t nb_dof)

Define number of DOF.

Parameters

<code>in,out</code>	<code>first_dof</code>	Label of the first DOF in input that is actualized
<code>in</code>	<code>nb_dof</code>	Number of DOF

void setCode (size_t dof, int code)

Assign code to a DOF.

Parameters

<code>in</code>	<code>dof</code>	DOF to which code is assigned
<code>in</code>	<code>code</code>	Code to assign

void setCode (const string & exp, int code, size_t dof = 1)

Define code by a boolean algebraic expression invoking coordinates of side nodes.

Parameters

<code>in</code>	<code>exp</code>	Boolean algebraic expression as required by <code>fparser</code>
<code>in</code>	<code>code</code>	Code to assign to node if the algebraic expression is true
<code>in</code>	<code>dof</code>	Degree of Freedom for which code is assigned [Default: 1]

void Add (Element * el)

Set pointer to neighbor element.

Parameters

<code>in</code>	<code>el</code>	Pointer to element to add as a neighbor element
-----------------	-----------------	---

Remarks

This function adds the pointer `e1` only if this one is not a null pointer

void set (Element * el, size_t i)

Set pointer to neighbor element.

Parameters

<code>in</code>	<code>el</code>	Pointer to element to set as a neighbor element
<code>in</code>	<code>i</code>	Local number of neighbor element

Remarks

This function differs from the `Add` by the fact that the local label of neighbor element is given

int getCode (size_t dof = 1) const

Return code for a given DOF of node.

Parameters

<code>in</code>	<code>dof</code>	Local label of degree of freedom. [Default: 1]
-----------------	------------------	--

Node* operator() (size_t i) const

Operator `()`.

Return pointer to node of local label `i`.

Element* **getNeighborElement** (**size_t i**) **const**

Return pointer to i-th side neighboring element.

Parameters

in	<i>i</i>	Local label of neighbor element (must be equal to 1 or 2)
-----------	----------	---

Element* getOtherNeighborElement (Element * el) const

Return pointer to other neighboring element than given one.

Parameters

in	<i>el</i>	Pointer to a given neighbor element
-----------	-----------	-------------------------------------

Remarks

If the side is on the boundary this function returns null pointer

Point<real_t> getNormal () const

Return normal vector to side.

The normal vector is oriented from the first neighbor element to the second one.

Warning

The norm of this vector is equal to the measure of the side (length of the edge in 2-D and area of the face in 3-D), and To get the unit normal, use rather the member function `getUnitNormal`.

Point<real_t> getUnitNormal () const

Return unit normal vector to side.

The unit normal vector is oriented from the first neighbor element to the second one.

Remarks

The norm of this vector is equal to one.

int isOnBoundary () const

Boundary side or not.

Returns 1 or -1 if side is on boundary Depending on whether the first or the second neighbor element is defined Returns 0 if side is an inner one

Remarks

This member function is valid only if member function `Mesh::getAllSides()` or `Mesh::getBoundarySides()` has been called before.

real_t getMeasure () const

Return measure of side.

This member function returns length or area of side. In case of quadrilaterals it returns determinant of Jacobian of mapping between reference and actual side

size_t Contains (const Node * nd) const

Say if a given node belongs to current side.

Parameters

<code>in</code>	<code>nd</code>	Pointer to searched node
-----------------	-----------------	--------------------------

Returns

index (local label) of node if found, 0 if not

void setChild (Side * sd)

Assign side as child of current one and assign current side as father This function is principally used when refining is invoked (e.g. for mesh adaption)

Parameters

<code>in</code>	<code>sd</code>	Pointer to side to assign
-----------------	-----------------	---------------------------

7.88 SideList Class Reference

Class to construct a list of sides having some common properties.

Public Member Functions

- [SideList](#) ([Mesh](#) &ms)
Constructor using a [Mesh](#) instance.
- [~SideList](#) ()
Destructor.
- void [selectCode](#) (int code, int dof=1)
Select sides having a given code for a given degree of freedom.
- void [unselectCode](#) (int code, int dof=1)
Unselect sides having a given code for a given degree of freedom.
- size_t [getNbSides](#) () const
Return number of selected sides.
- void [top](#) ()
Reset list of sides at its top position (Non constant version)
- void [top](#) () const
Reset list of sides at its top position (Constant version)
- [Side](#) * [get](#) ()
Return pointer to current side and move to next one (Non constant version)
- [Side](#) * [get](#) () const
Return pointer to current side and move to next one (Constant version)

7.88.1 Detailed Description

Class to construct a list of sides having some common properties.

This class enables choosing multiple selection criteria by using function `select...` However, the intersection of these properties must be empty.

7.88.2 Member Function Documentation

void selectCode (int code, int dof = 1)

Select sides having a given code for a given degree of freedom.

Parameters

in	code	Code that sides share
in	dof	Degree of Freedom label [Default: 1]

void unselectCode (int code, int dof = 1)

Unselect sides having a given code for a given degree of freedom.

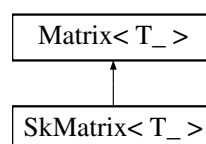
Parameters

in	code	Code of sides to exclude
in	dof	Degree of Freedom label [Default: 1]

7.89 SkMatrix< T_ > Class Template Reference

To handle square matrices in skyline storage format.

Inheritance diagram for SkMatrix< T_ >:



Public Member Functions

- [SkMatrix](#) ()
Default constructor.
- [SkMatrix](#) (size_t size, int is_diagonal=false)
Constructor that initializes a dense symmetric matrix.
- [SkMatrix](#) (class [Mesh](#) &mesh, size_t dof=0, int is_diagonal=false)
Constructor using mesh to initialize skyline structure of matrix.
- [SkMatrix](#) (const [Vect](#)< size_t > &ColHt)
Constructor that initializes skyline structure of matrix using vector of column heights.
- [SkMatrix](#) (const [SkMatrix](#)< T_ > &m)
Copy Constructor.
- [~SkMatrix](#) ()
Destructor.
- void [setMesh](#) (class [Mesh](#) &mesh, size_t dof=0)
Determine mesh graph and initialize matrix.
- void [setSkyline](#) (class [Mesh](#) &mesh)
Determine matrix structure.
- void [setDiag](#) ()
Store diagonal entries in a separate internal vector.
- void [setDOF](#) (size_t i)
Choose DOF to activate.
- void [set](#) (size_t i, size_t j, const T_ &val)
Assign a value to an entry of the matrix.

- void **Axpy** (T_ a, const **SkMatrix**< T_ > &m)
Add to matrix the product of a matrix by a scalar.
- void **Axpy** (T_ a, const **Matrix**< T_ > *m)
Add to matrix the product of a matrix by a scalar.
- void **MultAdd** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
Multiply matrix by vector x and add to y.
- void **TMultAdd** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
Multiply transpose of matrix by vector x and add to y.
- void **MultAdd** (T_ a, const **Vect**< T_ > &x, **Vect**< T_ > &y) const
Multiply matrix by a vector and add to another one.
- void **Mult** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
Multiply matrix by vector x and save in y.
- void **TMult** (const **Vect**< T_ > &x, **Vect**< T_ > &y) const
Multiply transpose of matrix by vector x and save in y.
- void **add** (size_t i, size_t j, const T_ &val)
Add a constant value to an entry of the matrix.
- size_t **getColHeight** (size_t i) const
Return column height.
- T_ **operator()** (size_t i, size_t j) const
Operator () (Constant version).
- T_ & **operator()** (size_t i, size_t j)
Operator () (Non constant version).
- void **DiagPrescribe** (class **Mesh** &mesh, **Vect**< T_ > &b, const **Vect**< T_ > &u, int flag=0)
Impose an essential boundary condition.
- void **DiagPrescribe** (**Vect**< T_ > &b, const **Vect**< T_ > &u, int flag=0)
Impose an essential boundary condition using the Mesh instance provided by the constructor.
- **SkMatrix**< T_ > & **operator=** (const **SkMatrix**< T_ > &m)
Operator =.
- **SkMatrix**< T_ > & **operator=** (const T_ &x)
Operator =.
- **SkMatrix**< T_ > & **operator+=** (const **SkMatrix**< T_ > &m)
Operator +=.
- **SkMatrix**< T_ > & **operator+=** (const T_ &x)
Operator +=.
- **SkMatrix**< T_ > & **operator*=** (const T_ &x)
*Operator *=.*
- int **Factor** ()
Factorize matrix (LU factorization).
- int **Solve** (**Vect**< T_ > &b)
Solve system with factorized matrix (forward and back substitution).
- int **Solve** (const **Vect**< T_ > &b, **Vect**< T_ > &x)
Solve system with factorized matrix (forward and back substitution).
- T_ * **get** () const
Return C-Array.
- T_ **get** (size_t i, size_t j) const
Return entry (i, j) of matrix if this one is stored, 0 else.

- size_t `getNbRows` () const
Return number of rows.
- size_t `getNbColumns` () const
Return number of columns.
- void `setPenal` (real_t p)
Set Penalty Parameter (For boundary condition prescription).
- void `setDiagonal` ()
Set the matrix as diagonal.
- void `setDiagonal` (const class `Mesh` &mesh)
Initialize matrix storage in the case where only diagonal terms are stored.
- T_ `getDiag` (size_t k) const
Return k-th diagonal entry of matrix.
- size_t `size` () const
Return matrix dimension (Number of rows and columns).
- void `Assembly` (const class `Element` *el, T_ *a)
Assembly of element matrix into global matrix.
- void `Assembly` (const class `Element` *el, const `DMatrix`< T_ > &a)
Assembly of element matrix into global matrix.
- void `Assembly` (const class `Side` *sd, T_ *a)
Assembly of side matrix into global matrix.
- void `Assembly` (const class `Side` *sd, const `DMatrix`< T_ > &a)
Assembly of side matrix into global matrix.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition.
- void `Prescribe` (int dof, int code, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- void `Prescribe` (`Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition, using the Mesh instance provided by the constructor.
- void `Prescribe` (const class `Mesh` &mesh, `Vect`< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition.
- void `Prescribe` (`Vect`< T_ > &b, int flag=0)
Impose by a penalty method a homogeneous (=0) essential boundary condition, using the Mesh instance provided by the constructor.
- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- int `FactorAndSolve` (`Vect`< T_ > &b)
Factorize matrix and solve the linear system.

- `int FactorAndSolve (const Vect< T_ > &b, Vect< T_ > &x)`
Factorize matrix and solve the linear system.
- `unsigned long getLength () const`
Return number of stored terms in matrix.
- `int isDiagonal () const`
Say if matrix is diagonal or not.
- `int isFactorized () const`
Say if matrix is factorized or not.
- `virtual size_t getColInd (size_t i) const`
Return Column index for column i (See the description for class `SpMatrix`).
- `virtual size_t getRowPtr (size_t i) const`
Return Row pointer for row i (See the description for class `SpMatrix`).
- `T_ operator() (size_t i) const`
Operator () with one argument (Constant version).
- `T_ & operator() (size_t i)`
Operator () with one argument (Non Constant version).
- `T_ & operator[] (size_t k)`
Operator [] (Non constant version).
- `T_ operator[] (size_t k) const`
Operator [] (Constant version).
- `Matrix & operator+= (const Matrix< T_ > &m)`
Operator +=.
- `Matrix & operator-= (const Matrix< T_ > &m)`
Operator -=.
- `Matrix & operator-= (const T_ &x)`
Operator -=.
- `void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)`
Set a window for matrix printing.
- `void getPrintView (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const`
Return window data for matrix printing.

7.89.1 Detailed Description

`template<class T_>class OFELI::SkMatrix< T_ >`

To handle square matrices in skyline storage format.

This template class allows storing and manipulating a matrix in skyline storage format.

The matrix entries are stored in 2 vectors column by column as in the following example:

```

/          \          /          \
| 10         . |      | u0  u1  0  0  u7 |
| 11 12         . |      |   u2  u3  0  u8 |
| 0 13 14         . |      | ...   u4  u5  u9 |
| 0 0 15 16         |      |           u6  u10 |
| 17 18 19 110 111 |      |           u11 |
\          /          \          /

```

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
-------------------------	---

7.89.2 Constructor & Destructor Documentation

SkMatrix ()

Default constructor.

Initializes a zero-dimension matrix

SkMatrix (size_t size, int is_diagonal = false)

Constructor that initializes a dense symmetric matrix.

Normally, for a dense matrix this is not the right class.

Parameters

<code>in</code>	<code>size</code>	Number of matrix rows (and columns).
<code>in</code>	<code>is_diagonal</code>	Boolean to select if the matrix is diagonal or not [Default: false]

SkMatrix (class Mesh & mesh, size_t dof = 0, int is_diagonal = false)

Constructor using mesh to initialize skyline structure of matrix.

Parameters

<code>in</code>	<code>mesh</code>	Mesh instance for which matrix graph is determined.
<code>in</code>	<code>dof</code>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
<code>in</code>	<code>is_diagonal</code>	Boolean argument to say is the matrix is actually a diagonal matrix or not.

SkMatrix (const Vect< size_t > & ColHt)

Constructor that initializes skyline structure of matrix using vector of column heights.

Parameters

<code>in</code>	<code>ColHt</code>	Vect instance that contains rows lengths of matrix.
-----------------	--------------------	---

7.89.3 Member Function Documentation

void setMesh (class Mesh & mesh, size_t dof = 0)

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

Parameters

<code>in</code>	<code>mesh</code>	Mesh instance for which matrix graph is determined.
<code>in</code>	<code>dof</code>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.

void setSkyline (class Mesh & mesh)

Determine matrix structure.

This member function calculates matrix structure using a [Mesh](#) instance.

Parameters

in	<i>mesh</i>	Mesh instance
----	-------------	-------------------------------

void setDOF (size_t i)

Choose DOF to activate.

This function is available only if variable dof is equal to 1 in the constructor

Parameters

in	<i>i</i>	Index of the DOF
----	----------	------------------

void set (size_t i, size_t j, const T_ & val) [virtual]

Assign a value to an entry of the matrix.

Parameters

in	<i>i</i>	Row index (starting at i=1)
in	<i>j</i>	Column index (starting at i=1)
in	<i>val</i>	Value to assign to entry a(i,j)

Implements [Matrix< T_ >](#).

void Axy (T_ a, const SkMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Matrix by which a is multiplied. The result is added to current instance

void Axy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Matrix by which a is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

void MultAdd (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and add to y.

Parameters

in	<i>x</i>	Vector to multiply by matrix
in,out	<i>y</i>	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void TMultAdd (const Vect< T_ > & x, Vect< T_ > & y) const

Multiply transpose of matrix by vector x and add to y.

Parameters

in	x	Vector to multiply by matrix
in,out	y	Vector to add to the result. y contains on output the result.

void MultAdd (T_ a, const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by a vector and add to another one.

Parameters

in	a	Constant to multiply by matrix
in	x	Vector to multiply by matrix
in,out	y	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void Mult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and save in y.

Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void TMult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply transpose of matrix by vector x and save in y.

Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void add (size_t i, size_t j, const T_ & val) [virtual]

Add a constant value to an entry of the matrix.

Parameters

in	i	Row index
in	j	Column index
in	val	Constant value to add to a(i,j)

Implements [Matrix< T_ >](#).

size_t getColHeight (size_t i) const

Return column height.

Column height at entry i is returned.

T_ operator() (size_t i, size_t j) const [virtual]

Operator () (Constant version).

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**T_ & operator() (size_t i, size_t j)** [virtual]

Operator () (Non constant version).

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**void DiagPrescribe (class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)**

Impose an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. It can be modified by member function **setPenal**(..).

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void DiagPrescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)Impose an essential boundary condition using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. It can be modified by member function **setPenal**(..).

Parameters

in	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

SkMatrix< T_ > & operator= (const SkMatrix< T_ > & m)

Operator =.

Copy matrix *m* to current matrix instance.

SkMatrix< T_ > & operator= (const T_ & x)

Operator =.

Assign constant value *x* to all matrix entries.

SkMatrix< T_ > & operator+= (const SkMatrix< T_ > & m)

Operator +=.

Add matrix *m* to current matrix instance.

SkMatrix< T_ > & operator+= (const T_ & x)

Operator +=.

Add constant value *x* to matrix entries.

SkMatrix< T_ > & operator*= (const T_ & x)

Operator *.

Premultiply matrix entries by constant value *x*.

int Factor () [virtual]

Factorize matrix (LU factorization).

Returns

- 0 if factorization was normally performed
- *n* if the *n*-th pivot is null

Implements [Matrix< T_ >](#).

int Solve (Vect< T_ > & b) [virtual]

Solve system with factorized matrix (forward and back substitution).

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output.
--------	----------	--

Returns

- 0 if solution was normally performed,
- *n* if the *n*-th pivot is null.

Solution is performed only if factorization has previously been invoked.

Implements [Matrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve system with factorized matrix (forward and back substitution).

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
----	----------	---

out	x	Vect instance that contains solution
-----	---	--

Returns

- 0 if solution was normally performed,
 - *n* if the *n*-th pivot is null
- Solution is performed only if factorization has previously been invoked.

T_* get () const

Return C-Array.

Skyline of matrix is stored row by row.

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return *k*-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This gunction is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in	<i>mesh</i>	Mesh instance from which information is extracted.

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
--------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.

Matrix< T_ > & operator-= (const Matrix< T_ > & m) [inherited]

Operator -=.

Subtract matrix *m* from current matrix instance.

Matrix< T_ > & operator-= (const T_ & x) [inherited]

Operator -=.

Subtract constant value x from all matrix entries.

void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax) [inherited]

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const [inherited]

Return window data for matrix printing.

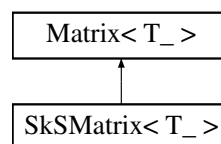
Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.90 SkSMatrix< T_ > Class Template Reference

To handle symmetric matrices in skyline storage format.

Inheritance diagram for SkSMatrix< T_ >:



Public Member Functions

- [SkSMatrix](#) ()
Default constructor.
- [SkSMatrix](#) (size_t [size](#), int is_diagonal=false)
Constructor that initializes a dense symmetric matrix.
- [SkSMatrix](#) (class [Mesh](#) &mesh, size_t dof=0, int is_diagonal=false)
Constructor using mesh to initialize skyline structure of matrix.
- [SkSMatrix](#) (const [Vect](#)< size_t > &ColHt)
Constructor that initializes skyline structure of matrix using vector of column height.
- [SkSMatrix](#) (const [Vect](#)< size_t > &I, const [Vect](#)< size_t > &J, int opt=1)
Constructor for a square matrix using non zero row and column indices.
- [SkSMatrix](#) (const [Vect](#)< size_t > &I, const [Vect](#)< size_t > &J, const [Vect](#)< T_ > &a, int opt=1)

- Constructor for a square matrix using non zero row and column indices.*

 - `SkSMatrix` (const `SkSMatrix`< T_ > &m)
- Copy Constructor.*

 - `~SkSMatrix` ()
- Destructor.*

 - void `setMesh` (class `Mesh` &mesh, size_t dof=0)
- Determine mesh graph and initialize matrix.*

 - void `setSkyline` (class `Mesh` &mesh)
- Determine matrix structure.*

 - void `setDiag` ()
- Store diagonal entries in a separate internal vector.*

 - void `set` (size_t i, size_t j, const T_ &val)
- Assign a value to an entry of the matrix.*

 - void `Axpy` (T_ a, const `SkSMatrix`< T_ > &m)
- Add to matrix the product of a matrix by a scalar.*

 - void `Axpy` (T_ a, const `Matrix`< T_ > *m)
- Add to matrix the product of a matrix by a scalar.*

 - void `MultAdd` (const `Vect`< T_ > &x, `Vect`< T_ > &y) const
- Multiply matrix by vector x and add to y.*

 - void `MultAdd` (T_ a, const `Vect`< T_ > &x, `Vect`< T_ > &y) const
- Multiply matrix by vector a*x and add to y.*

 - void `Mult` (const `Vect`< T_ > &x, `Vect`< T_ > &y) const
- Multiply matrix by vector x and save in y*

 - void `TMult` (const `Vect`< T_ > &x, `Vect`< T_ > &y) const
- Multiply transpose of matrix by vector x and save in y.*

 - void `add` (size_t i, size_t j, const T_ &val)
- Add a constant to an entry of the matrix.*

 - size_t `getColHeight` (size_t i) const
- Return column height.*

 - `Vect`< T_ > `getColumn` (size_t j) const
- Get j-th column vector.*

 - `Vect`< T_ > `getRow` (size_t i) const
- Get i-th row vector.*

 - T_ & `operator()` (size_t i, size_t j)
- Operator () (Non constant version).*

 - T_ `operator()` (size_t i, size_t j) const
- Operator () (Constant version).*

 - `SkSMatrix`< T_ > & `operator=` (const `SkSMatrix`< T_ > &m)
- Operator =.*

 - `SkSMatrix`< T_ > & `operator=` (const T_ &x)
- Operator =.*

 - `SkSMatrix`< T_ > & `operator+=` (const `SkSMatrix`< T_ > &m)
- Operator +=.*

 - `SkSMatrix`< T_ > & `operator*=` (const T_ &x)
- Operator *.=.*

 - int `Factor` ()

- Factorize matrix (LDLt factorization).*
- int [Solve](#) ([Vect](#)< T_ > &b)
 - Solve system with factorized matrix (forward and back substitution).*
- int [Solve](#) (const [Vect](#)< T_ > &b, [Vect](#)< T_ > &x)
 - Solve system with factorized matrix (forward and back substitution).*
- T_ * [get](#) () const
 - Return C-Array.*
- void [set](#) (size_t i, T_ x)
 - Assign a value to the i-th entry of C-array containing matrix.*
- T_ [get](#) (size_t i, size_t j) const
 - Return entry (i, j) of matrix if this one is stored, 0 else.*
- size_t [getNbRows](#) () const
 - Return number of rows.*
- size_t [getNbColumns](#) () const
 - Return number of columns.*
- void [setPenal](#) (real_t p)
 - Set Penalty Parameter (For boundary condition prescription).*
- void [setDiagonal](#) ()
 - Set the matrix as diagonal.*
- void [setDiagonal](#) (const class [Mesh](#) &mesh)
 - Initialize matrix storage in the case where only diagonal terms are stored.*
- T_ [getDiag](#) (size_t k) const
 - Return k-th diagonal entry of matrix.*
- size_t [size](#) () const
 - Return matrix dimension (Number of rows and columns).*
- void [Assembly](#) (const class [Element](#) *el, T_ *a)
 - Assembly of element matrix into global matrix.*
- void [Assembly](#) (const class [Element](#) *el, const [DMatrix](#)< T_ > &a)
 - Assembly of element matrix into global matrix.*
- void [Assembly](#) (const class [Side](#) *sd, T_ *a)
 - Assembly of side matrix into global matrix.*
- void [Assembly](#) (const class [Side](#) *sd, const [DMatrix](#)< T_ > &a)
 - Assembly of side matrix into global matrix.*
- void [Prescribe](#) (const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
 - Impose by a penalty method an essential boundary condition.*
- void [Prescribe](#) (int dof, int code, const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
 - Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.*
- void [Prescribe](#) ([Vect](#)< T_ > &b, const [Vect](#)< T_ > &u, int flag=0)
 - Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.*
- void [Prescribe](#) (const class [Mesh](#) &mesh, [Vect](#)< T_ > &b, int flag=0)
 - Impose by a penalty method a homogeneous (=0) essential boundary condition.*
- void [Prescribe](#) ([Vect](#)< T_ > &b, int flag=0)
 - Impose by a penalty method a homogeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.*

- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- int `FactorAndSolve` (`Vect`< T_ > &b)
Factorize matrix and solve the linear system.
- int `FactorAndSolve` (const `Vect`< T_ > &b, `Vect`< T_ > &x)
Factorize matrix and solve the linear system.
- unsigned long `getLength` () const
Return number of stored terms in matrix.
- int `isDiagonal` () const
Say if matrix is diagonal or not.
- int `isFactorized` () const
Say if matrix is factorized or not.
- virtual size_t `getColInd` (size_t i) const
Return Column index for column i (See the description for class `SpMatrix`).
- virtual size_t `getRowPtr` (size_t i) const
Return Row pointer for row i (See the description for class `SpMatrix`).
- T_ `operator()` (size_t i) const
Operator () with one argument (Constant version).
- T_ & `operator()` (size_t i)
Operator () with one argument (Non Constant version).
- T_ & `operator[]` (size_t k)
Operator [] (Non constant version).
- T_ `operator[]` (size_t k) const
Operator [] (Constant version).
- `Matrix` & `operator+=` (const `Matrix`< T_ > &m)
Operator +=.
- `Matrix` & `operator+=` (const T_ &x)
Operator +=.
- `Matrix` & `operator-=` (const `Matrix`< T_ > &m)
Operator -=.
- `Matrix` & `operator-=` (const T_ &x)
Operator -=.
- void `setPrintView` (size_t rmin, size_t rmax, size_t cmin, size_t cmax)
Set a window for matrix printing.
- void `getPrintView` (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
Return window data for matrix printing.

7.90.1 Detailed Description

template<class T_>class OFELI::SkSMatrix< T_ >

To handle symmetric matrices in skyline storage format.
This template class allows storing and manipulating a symmetric matrix in skyline storage format.
The matrix entries are stored column by column as in the following example:

```

/
| a0  a1  0  0  a7 |
|      a2  a3  0  a8 |
| ...      a4  a5  a9 |
|              a6  a10 |
|                  a11 |
\

```

Template Parameters

<code><T_></code>	Data type (double, float, complex<double>, ...)
-------------------------	---

7.90.2 Constructor & Destructor Documentation

SkSMatrix ()

Default constructor.
Initializes a zero-dimension matrix

SkSMatrix (size_t size, int is_diagonal = false)

Constructor that initializes a dense symmetric matrix.
Normally, for a dense matrix this is not the right class.
Parameters

<code>in</code>	<code>size</code>	Number of matrix rows (and columns).
<code>in</code>	<code>is_diagonal</code>	Boolean to select if the matrix is diagonal or not [Default: false]

SkSMatrix (class Mesh & mesh, size_t dof = 0, int is_diagonal = false)

Constructor using mesh to initialize skyline structure of matrix.
Parameters

<code>in</code>	<code>mesh</code>	Mesh instance for which matrix graph is determined.
<code>in</code>	<code>dof</code>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
<code>in</code>	<code>is_diagonal</code>	Boolean argument to say is the matrix is actually a diagonal matrix or not.

SkSMatrix (const Vect< size_t > & ColHt)

Constructor that initializes skyline structure of matrix using vector of column height.

Parameters

<code>in</code>	<code>ColHt</code>	Vect instance that contains rows lengths of matrix.
-----------------	--------------------	---

SkSMatrix (const Vect< size_t > & I, const Vect< size_t > & J, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

Parameters

<code>in</code>	<code>I</code>	Vector containing row indices
<code>in</code>	<code>J</code>	Vector containing column indices
<code>in</code>	<code>opt</code>	Flag indicating if vectors I and J are cleaned and ordered (opt=1) or not (opt=0). In the latter case, these vectors can contain the same contents more than once and are not necessarily ordered.

SkSMatrix (const Vect< size_t > & I, const Vect< size_t > & J, const Vect< T_ > & a, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

Parameters

<code>in</code>	<code>I</code>	Vector containing row indices
<code>in</code>	<code>J</code>	Vector containing column indices
<code>in</code>	<code>a</code>	Vector containing matrix entries in the same order than the one given by I and J
<code>in</code>	<code>opt</code>	Flag indicating if vectors I and J are cleaned and ordered (opt=1) or not (opt=0). In the latter case, these vectors can contain the same contents more than once and are not necessarily ordered

7.90.3 Member Function Documentation

void setMesh (class Mesh & mesh, size_t dof = 0)

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

Parameters

<code>in</code>	<code>mesh</code>	Mesh instance for which matrix graph is determined.
<code>in</code>	<code>dof</code>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.

void setSkyline (class Mesh & mesh)

Determine matrix structure.

This member function calculates matrix structure using [Mesh](#) instance mesh.

void set (size_t i, size_t j, const T_ & val) [virtual]

Assign a value to an entry of the matrix.

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index
in	<i>val</i>	Value to assign to a(i,j)

Implements [Matrix< T_ >](#).

void Apxy (T_ a, const SkSMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Matrix by which a is multiplied. The result is added to current instance

void Apxy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Pointer to Matrix by which a is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

void MultAdd (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and add to y.

Parameters

in	<i>x</i>	Vector to multiply by matrix
in,out	<i>y</i>	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void MultAdd (T_ a, const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector a*x and add to y.

Parameters

in	<i>a</i>	Constant to multiply by matrix
in	<i>x</i>	Vector to multiply by matrix
in,out	<i>y</i>	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void Mult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and save in y

Parameters

in	<i>x</i>	Vector to multiply by matrix
out	<i>y</i>	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void TMult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply transpose of matrix by vector x and save in y.

Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void add (size_t i, size_t j, const T_ & val) [virtual]

Add a constant to an entry of the matrix.

Parameters

in	i	Row index
in	j	Column index
in	val	Constant value to add to a(i,j)

Implements [Matrix< T_ >](#).

size_t getColHeight (size_t i) const

Return column height.

Column height at entry i is returned.

T_ & operator() (size_t i, size_t j) [virtual]

Operator () (Non constant version).

Parameters

in	i	Row index
in	j	Column index

Implements [Matrix< T_ >](#).

T_ operator() (size_t i, size_t j) const [virtual]

Operator () (Constant version).

Parameters

in	i	Row index
in	j	Column index

Implements [Matrix< T_ >](#).

SkSMatrix< T_ > & operator= (const SkSMatrix< T_ > & m)

Operator =.

Copy matrix m to current matrix instance.

SkSMatrix< T_ > & operator= (const T_ & x)

Operator =.

Assign constant value x to all matrix entries.

SkSMatrix< T_ > & operator+= (const SkSMatrix< T_ > & m)

Operator +=.

Add matrix m to current matrix instance.

SkSMatrix< T_ > & operator*= (const T_ & x)

Operator *.=.

Premultiply matrix entries by constant value x.

int Factor () [virtual]

Factorize matrix (LDLt factorization).

Returns

0 if factorization was normally performed,
n if the n-th pivot is null.

Implements [Matrix< T_ >](#).

int Solve (Vect< T_ > & b) [virtual]

Solve system with factorized matrix (forward and back substitution).

Parameters

in,out	b	Vect instance that contains right-hand side on input and solution on output.
--------	---	--

Returns

0, if solution was normally performed, n if the n-th pivot is null. Solution is performed only if factorization has previously been invoked.

Implements [Matrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve system with factorized matrix (forward and back substitution).

Parameters

in	b	Vect instance that contains right-hand side
out	x	Vect instance that contains solution

Returns

0 if solution was normally performed, n if the n-th pivot is null. Solution is performed only if factorization has previously been invoked.

T_* get () const

Return C-Array.

Skyline of matrix is stored row by row.

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return k-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.

<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).
-----------	-------------	--

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
<i>in</i>	<i>code</i>	Code for which a boundary condition is to be enforced
<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homogeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
--------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

<i>in</i>	<i>i</i>	entry index
-----------	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns k-th stored element in matrix Index k starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns k-th stored element in matrix Index k starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.**Matrix< T_ > & operator+= (const T_ & x)** [inherited]

Operator +=.

Add constant value *x* to all matrix entries.**Matrix< T_ > & operator-= (const Matrix< T_ > & m)** [inherited]

Operator -=.

Subtract matrix *m* from current matrix instance.**Matrix< T_ > & operator-= (const T_ & x)** [inherited]

Operator -=.

Subtract constant value *x* from all matrix entries.**void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)** [inherited]

Set a window for matrix printing.

Parameters

<i>in</i>	<i>rmin</i>	first row index to view
<i>in</i>	<i>rmax</i>	last row index to view
<i>in</i>	<i>cmin</i>	first column index to view
<i>in</i>	<i>cmax</i>	last column index to view

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const [inherited]

Return window data for matrix printing.

Parameters

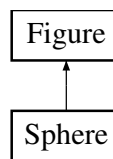
<i>out</i>	<i>rmin</i>	first row index to view
------------	-------------	-------------------------

out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.91 Sphere Class Reference

To store and treat a sphere.

Inheritance diagram for Sphere:



Public Member Functions

- [Sphere](#) ()
Default construcor.
- [Sphere](#) (const [Point](#)< double > &c, double r, int code=1)
Constructor.
- void [setRadius](#) (double r)
Assign radius of sphere.
- double [getRadius](#) () const
Return radius of sphere.
- void [setCenter](#) (const [Point](#)< double > &c)
Assign coordinates of center of sphere.
- [Point](#)< double > [getCenter](#) () const
Return coordinates of center of sphere.
- double [getSignedDistance](#) (const [Point](#)< double > &p) const
Return signed distance of a given point from the current sphere.
- [Sphere](#) & [operator+=](#) ([Point](#)< double > a)
Operator +=.
- [Sphere](#) & [operator*=](#) (double a)
*Operator *.=.*
- void [setCode](#) (int code)
Choose a code for the domain defined by the figure.
- void [getSignedDistance](#) (const [Grid](#) &g, [Vect](#)< double > &d) const
Calculate signed distance to current figure with respect to grid points.
- double [dLine](#) (const [Point](#)< double > &p, const [Point](#)< double > &a, const [Point](#)< double > &b) const
Compute signed distance from a line.

7.91.1 Detailed Description

To store and treat a sphere.

7.91.2 Constructor & Destructor Documentation

Sphere (const Point< double > & c, double r, int code = 1)

Constructor.

Parameters

in	<i>c</i>	Coordinates of center of sphere
in	<i>r</i>	Radius
in	<i>code</i>	Code to assign to the generated sphere [Default: 1]

7.91.3 Member Function Documentation

double getSignedDistance (const Point< double > & p) const [virtual]

Return signed distance of a given point from the current sphere.

The computed distance is negative if p lies in the ball, positive if it is outside, and 0 on the sphere

Parameters

in	<i>p</i>	Point<double> instance
----	----------	------------------------

Reimplemented from [Figure](#).

Sphere& operator+= (Point< double > a)

Operator +=.

Translate sphere by a vector a

Sphere& operator+= (double a)

Operator *.=.

Scale sphere by a factor a

void getSignedDistance (const Grid & g, Vect< double > & d) const [inherited]

Calculate signed distance to current figure with respect to grid points.

Parameters

in	<i>g</i>	Grid instance
in	<i>d</i>	Vect instance containing calculated distance from each grid index to Figure

Remarks

Vector d doesn't need to be sized before invoking this function

double dLine (const Point< double > & p, const Point< double > & a, const Point< double > & b) const [inherited]

Compute signed distance from a line.

Parameters

in	<i>p</i>	Point for which distance is computed
in	<i>a</i>	First vertex of line
in	<i>b</i>	Second vertex of line

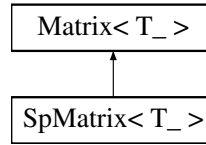
Returns

Signed distance

7.92 SpMatrix< T_ > Class Template Reference

To handle matrices in sparse storage format.

Inheritance diagram for SpMatrix< T_ >:



Public Member Functions

- [SpMatrix](#) ()
Default constructor.
- [SpMatrix](#) (size_t nr, size_t nc)
Constructor that initializes current instance as a dense matrix.
- [SpMatrix](#) (size_t size, int is_diagonal=false)
Constructor that initializes current instance as a dense matrix.
- [SpMatrix](#) (class [Mesh](#) &mesh, size_t dof=0, int is_diagonal=false)
Constructor using a [Mesh](#) instance.
- [SpMatrix](#) (const class vector< std::pair< size_t, size_t > > &l, int opt=1)
Constructor for a square matrix using non zero row and column indices.
- [SpMatrix](#) (const class vector< std::pair< size_t, size_t > > &l, const class vector< T_ > &a, int opt=1)
Constructor for a square matrix using non zero row and column indices.
- [SpMatrix](#) (size_t nr, size_t nc, const vector< size_t > &row_ptr, const vector< size_t > &col_ind)
Constructor for a rectangle matrix.
- [SpMatrix](#) (size_t nr, size_t nc, const vector< size_t > &row_ptr, const vector< size_t > &col_ind, const vector< T_ > &a)
Constructor for a rectangle matrix.
- [SpMatrix](#) (const vector< size_t > &row_ptr, const vector< size_t > &col_ind)
Constructor for a rectangle matrix.
- [SpMatrix](#) (const vector< size_t > &row_ptr, const vector< size_t > &col_ind, const vector< T_ > &a)
Constructor for a rectangle matrix.
- [SpMatrix](#) (const [SpMatrix](#) &m)
Copy constructor.
- [~SpMatrix](#) (void)
Destructor.
- void [setMesh](#) (class [Mesh](#) &mesh, size_t dof=0)
Determine mesh graph and initialize matrix.
- void [setOneDOF](#) ()
Activate 1-DOF per node option.
- void [setSides](#) ()
Activate Sides option.
- void [setDiag](#) ()
Store diagonal entries in a separate internal vector.

- `Vect< T_ > getColumn (size_t j) const`
Get j-th column vector.
- `void DiagPrescribe (class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u)`
Impose by a diagonal method an essential boundary condition.
- `void DiagPrescribe (Vect< T_ > &b, const class Vect< T_ > &u)`
Impose by a diagonal method an essential boundary condition using the Mesh instance provided by the constructor.
- `void setSize (size_t size)`
Set size of matrix (case where it's a square matrix).
- `void setSize (size_t nr, size_t nc)`
Set size (number of rows) of matrix.
- `void setGraph (const vector< std::pair< size_t, size_t > > &l, int opt=1)`
Set graph of matrix by giving a vector of its nonzero entries.
- `Vect< T_ > getRow (size_t i) const`
Get i-th row vector.
- `T_ & operator() (size_t i, size_t j)`
Operator () (Non constant version)
- `T_ operator() (size_t i, size_t j) const`
Operator () (Constant version)
- `const T_ operator() (size_t i) const`
Operator () with one argument (Constant version)
- `const T_ operator[] (size_t i) const`
Operator [] (Constant version).
- `void getMesh (class Mesh &mesh)`
Get mesh instance whose reference will be stored in current instance of SpMatrix.
- `void Mult (const Vect< T_ > &v, Vect< T_ > &w) const`
Multiply matrix by vector and save in another one.
- `void MultAdd (const Vect< T_ > &x, Vect< T_ > &y) const`
Multiply matrix by vector x and add to y.
- `void MultAdd (T_ a, const Vect< T_ > &x, Vect< T_ > &y) const`
*Multiply matrix by vector a*x and add to y.*
- `void TMult (const Vect< T_ > &x, Vect< T_ > &y) const`
Multiply transpose of matrix by vector x and save in y.
- `void Axy (T_ a, const SpMatrix< T_ > &m)`
Add to matrix the product of a matrix by a scalar.
- `void Axy (T_ a, const Matrix< T_ > *m)`
Add to matrix the product of a matrix by a scalar.
- `void set (size_t i, size_t j, const T_ &val)`
Assign a value to an entry of the matrix.
- `void add (size_t i, size_t j, const T_ &val)`
Add a value to an entry of the matrix.
- `void operator= (const T_ &x)`
Operator =.
- `size_t getColInd (size_t i) const`
Return storage information.
- `size_t getRowPtr (size_t i) const`
Return Row pointer at position i.

- `int Solve (class Vect< T_ > &b)`
Solve the linear system of equations.
- `int Solve (const Vect< T_ > &b, Vect< T_ > &x)`
Solve the linear system of equations.
- `void setSolver (Iteration solver=CG_SOLVER, Preconditioner prec=DIAG_PREC, int max_it=1000, real_t toler=1.e-8)`
Choose solver and preconditioner for an iterative procedure.
- `T_ * get () const`
Return C-Array.
- `T_ get (size_t i, size_t j) const`
Return entry (i, j) of matrix if this one is stored, 0 otherwise.
- `size_t getNbRows () const`
Return number of rows.
- `size_t getNbColumns () const`
Return number of columns.
- `void setPenal (real_t p)`
Set Penalty Parameter (For boundary condition prescription).
- `void setDiagonal ()`
Set the matrix as diagonal.
- `void setDiagonal (const class Mesh &mesh)`
Initialize matrix storage in the case where only diagonal terms are stored.
- `T_ getDiag (size_t k) const`
Return k-th diagonal entry of matrix.
- `size_t size () const`
Return matrix dimension (Number of rows and columns).
- `void Assembly (const class Element *el, T_ *a)`
Assembly of element matrix into global matrix.
- `void Assembly (const class Element *el, const DMatrix< T_ > &a)`
Assembly of element matrix into global matrix.
- `void Assembly (const class Side *sd, T_ *a)`
Assembly of side matrix into global matrix.
- `void Assembly (const class Side *sd, const DMatrix< T_ > &a)`
Assembly of side matrix into global matrix.
- `void Prescribe (const class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
Impose by a penalty method an essential boundary condition.
- `void Prescribe (int dof, int code, const class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.
- `void Prescribe (Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
Impose by a penalty method an essential boundary condition, using the Mesh instance provided by the constructor.
- `void Prescribe (const class Mesh &mesh, Vect< T_ > &b, int flag=0)`
Impose by a penalty method a homogeneous (=0) essential boundary condition.
- `void Prescribe (Vect< T_ > &b, int flag=0)`
Impose by a penalty method a homogeneous (=0) essential boundary condition, using the Mesh instance provided by the constructor.

- void `Prescribe` (size_t dof, const class `Mesh` &mesh, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `Prescribe` (size_t dof, `Vect`< T_ > &b, const `Vect`< T_ > &u, int flag=0)
Impose by a penalty method an essential boundary condition when only one DOF is treated.
- void `PrescribeSide` (const class `Mesh` &mesh)
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- void `PrescribeSide` ()
Impose by a penalty method an essential boundary condition when DOFs are supported by sides.
- virtual int `Factor` ()=0
Factorize matrix. Available only if the storage class enables it.
- int `FactorAndSolve` (`Vect`< T_ > &b)
Factorize matrix and solve the linear system.
- int `FactorAndSolve` (const `Vect`< T_ > &b, `Vect`< T_ > &x)
Factorize matrix and solve the linear system.
- unsigned long `getLength` () const
Return number of stored terms in matrix.
- int `isDiagonal` () const
Say if matrix is diagonal or not.
- int `isFactorized` () const
Say if matrix is factorized or not.
- T_ & `operator`() (size_t i)
Operator () with one argument (Non Constant version).
- T_ & `operator`[] (size_t k)
Operator [] (Non constant version).
- `Matrix` & `operator+=` (const `Matrix`< T_ > &m)
Operator +=.
- `Matrix` & `operator+=` (const T_ &x)
Operator +=.
- `Matrix` & `operator-=` (const `Matrix`< T_ > &m)
Operator -=.
- `Matrix` & `operator-=` (const T_ &x)
Operator -=.
- `Matrix` & `operator*=` (const T_ &x)
*Operator *=.*
- void `setPrintView` (size_t rmin, size_t rmax, size_t cmin, size_t cmax)
Set a window for matrix printing.
- void `getPrintView` (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
Return window data for matrix printing.

7.92.1 Detailed Description

template<class T_>class OFELI::SpMatrix< T_ >

To handle matrices in sparse storage format.

This template class enables to store and manipulate a sparse matrix, i.e. only nonzero terms are stored. Internally, the matrix is stored as a vector instance and uses for the definition of its graph a `Vect`<size_t> instance `row_ptr` and a `Vect`<size_t> instance `col_ind` that contains respectively addresses of first element of each row and column indices.

To illustrate this, consider the matrix

```

1  2  0
3  4  0
0  5  0

```

Such a matrix is stored in the `vector<real_t>` instance `{1,2,3,4,5}`. The vectors `row_ptr` and `col_ind` are respectively: `{1,3,5,6}`, `{1,2,1,2,2}`

Template Parameters

<code><T_></code>	Data type (<code>real_t</code> , <code>float</code> , <code>complex<real_t></code> , ...)
-------------------------	--

7.92.2 Constructor & Destructor Documentation

SpMatrix ()

Default constructor.

Initialize a zero-dimension matrix

SpMatrix (size_t nr, size_t nc)

Constructor that initializes current instance as a dense matrix.

Normally, for a dense matrix this is not the right class.

Parameters

<code>in</code>	<code>nr</code>	Number of matrix rows.
<code>in</code>	<code>nc</code>	Number of matrix columns.

SpMatrix (size_t size, int is_diagonal = false)

Constructor that initializes current instance as a dense matrix.

Normally, for a dense matrix this is not the right class.

Parameters

<code>in</code>	<code>size</code>	Number of matrix rows (and columns).
<code>in</code>	<code>is_diagonal</code>	Boolean argument to say is the matrix is actually a diagonal matrix or not.

SpMatrix (class Mesh & mesh, size_t dof = 0, int is_diagonal = false)

Constructor using a [Mesh](#) instance.

Parameters

<code>in</code>	<code>mesh</code>	Mesh instance from which matrix graph is extracted.
<code>in</code>	<code>dof</code>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.
<code>in</code>	<code>is_diagonal</code>	Boolean argument to say is the matrix is actually a diagonal matrix or not.

SpMatrix (const class vector< std::pair< size_t, size_t > > & I, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

Parameters

in	<i>I</i>	Vector containing pairs of row and column indices
in	<i>opt</i>	Flag indicating if vectors <i>I</i> is cleaned and ordered (<i>opt</i> =1) or not (<i>opt</i> =0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

SpMatrix (const class vector< std::pair< size_t, size_t > > & I, const class vector< T_ > & a, int opt = 1)

Constructor for a square matrix using non zero row and column indices.

Parameters

in	<i>I</i>	Vector containing pairs of row and column indices
in	<i>a</i>	Vector containing matrix entries in the same order than the one given by <i>I</i>
in	<i>opt</i>	Flag indicating if vector <i>I</i> is cleaned and ordered (<i>opt</i> =1: default) or not (<i>opt</i> =0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

SpMatrix (size_t nr, size_t nc, const vector< size_t > & row_ptr, const vector< size_t > & col_ind)

Constructor for a rectangle matrix.

Parameters

in	<i>nr</i>	Number of rows
in	<i>nc</i>	Number of columns
in	<i>row_ptr</i>	Vector of row pointers (See the above description of this class).
in	<i>col_ind</i>	Vector of column indices (See the above description of this class).

SpMatrix (size_t nr, size_t nc, const vector< size_t > & row_ptr, const vector< size_t > & col_ind, const vector< T_ > & a)

Constructor for a rectangle matrix.

Parameters

in	<i>nr</i>	Number of rows
in	<i>nc</i>	Number of columns
in	<i>row_ptr</i>	Vector of row pointers (See the above description of this class).
in	<i>col_ind</i>	Vector of column indices (See the above description of this class).
in	<i>a</i>	vector instance containing matrix entries stored columnwise

SpMatrix (const vector< size_t > & row_ptr, const vector< size_t > & col_ind)

Constructor for a rectangle matrix.

Parameters

in	<i>row_ptr</i>	Vector of row pointers (See the above description of this class).
in	<i>col_ind</i>	Vector of column indices (See the above description of this class).

SpMatrix (const vector< size_t > & row_ptr, const vector< size_t > & col_ind, const vector< T_ > & a)

Constructor for a rectangle matrix.

Parameters

in	<i>row_ptr</i>	Vector of row pointers (See the above description of this class).
in	<i>col_ind</i>	Vector of column indices (See the above description of this class).
in	<i>a</i>	vector instance that contain matrix entries stored row by row. Number of rows is extracted from vector <i>row_ptr</i> .

7.92.3 Member Function Documentation

void setMesh (class Mesh & mesh, size_t dof = 0)

Determine mesh graph and initialize matrix.

This member function is called by constructor with the same arguments

Parameters

in	<i>mesh</i>	Mesh instance for which matrix graph is determined.
in	<i>dof</i>	Option parameter, with default value 0. dof=1 means that only one degree of freedom for each node (or element or side) is taken to determine matrix structure. The value dof=0 means that matrix structure is determined using all DOFs.

void DiagPrescribe (class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u)

Impose by a diagonal method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.

void DiagPrescribe (Vect< T_ > & b, const class Vect< T_ > & u)

Impose by a diagonal method an essential boundary condition using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.

void setSize (size_t size)

Set size of matrix (case where it's a square matrix).

Parameters

in	<i>size</i>	Number of rows and columns.
-----------	-------------	-----------------------------

void setSize (size_t nr, size_t nc)

Set size (number of rows) of matrix.

Parameters

in	<i>nr</i>	Number of rows
in	<i>nc</i>	Number of columns

void setGraph (const vector< std::pair< size_t, size_t > > & I, int opt = 1)

Set graph of matrix by giving a vector of its nonzero entries.

Parameters

in	<i>I</i>	Vector containing pairs of row and column indices
in	<i>opt</i>	Flag indicating if vector I is cleaned and ordered (opt=1: default) or not (opt=0). In the latter case, this vector can have the same contents more than once and are not necessarily ordered

T_& operator() (size_t i, size_t j) [virtual]

Operator () (Non constant version)

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**T_ operator() (size_t i, size_t j) const [virtual]**

Operator () (Constant version)

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**const T_ operator() (size_t i) const**

Operator () with one argument (Constant version)

Returns i-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

const T_ operator[] (size_t i) const

Operator [] (Constant version).

Returns i-th position in the array storing matrix entries. The first entry is at location 0. Entries are stored row by row.

void Mult (const Vect< T_ > & v, Vect< T_ > & w) const [virtual]

Multiply matrix by vector and save in another one.

Parameters

in	v	Vector to multiply by matrix
out	w	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void MultAdd (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector x and add to y.

Parameters

in	x	Vector to multiply by matrix
out	y	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void MultAdd (T_ a, const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply matrix by vector a*x and add to y.

Parameters

in	a	Constant to multiply by matrix
in	x	Vector to multiply by matrix
out	y	Vector to add to the result. y contains on output the result.

Implements [Matrix< T_ >](#).

void TMult (const Vect< T_ > & x, Vect< T_ > & y) const [virtual]

Multiply transpose of matrix by vector x and save in y.

Parameters

in	x	Vector to multiply by matrix
out	y	Vector that contains on output the result.

Implements [Matrix< T_ >](#).

void Axy (T_ a, const SpMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

in	a	Scalar to premultiply
in	m	Matrix by which a is multiplied. The result is added to current instance

void Axy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

in	<i>a</i>	Scalar to premultiply
in	<i>m</i>	Pointer to Matrix by which <i>a</i> is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).

void set (size_t i, size_t j, const T_ & val) [virtual]

Assign a value to an entry of the matrix.

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index
in	<i>val</i>	Value to assign to <i>a</i> (<i>i</i> , <i>j</i>)

Implements [Matrix< T_ >](#).

void add (size_t i, size_t j, const T_ & val) [virtual]

Add a value to an entry of the matrix.

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index
in	<i>val</i>	Constant value to add to <i>a</i> (<i>i</i> , <i>j</i>)

Implements [Matrix< T_ >](#).

void operator= (const T_ & x)

Operator =.

Assign constant value *x* to all matrix entries.

size_t getColInd (size_t i) const [virtual]

Return storage information.

Returns

Column index of the *i*-th stored element in matrix

Reimplemented from [Matrix< T_ >](#).

int Solve (class Vect< T_ > & b) [virtual]

Solve the linear system of equations.

The default parameters are:

- CG_SOLVER for solver
- DIAG_PREC for preconditioner
- Max. Number of iterations is 1000
- Tolerance is 1.e-8

To change these values, call function `setSolver` before this function

Parameters

in,out	<i>b</i>	Vector that contains right-hand side on input and solution on output
--------	----------	--

Returns

Number of actual performed iterations

Implements [Matrix< T_ >](#).

int Solve (const Vect< T_ > & b, Vect< T_ > & x)

Solve the linear system of equations.

The default parameters are:

- CG_SOLVER for solver
- DIAG_PREC for preconditioner
- Max. Number of iterations is 1000
- Tolerance is 1.e-8

To change these values, call function setSolver before this function

Parameters

in	<i>b</i>	Vector that contains right-hand side
out	<i>x</i>	Vector that contains the obtained solution

Returns

Number of actual performed iterations

void setSolver (Iteration solver = CG_SOLVER, Preconditioner prec = DIAG_PREC, int max_it = 1000, real_t toler = 1.e-8)

Choose solver and preconditioner for an iterative procedure.

Parameters

in	<i>solver</i>	Option to choose iterative solver in an enumerated variable <ul style="list-style-type: none"> • CG_SOLVER: Conjugate Gradient [default] • CGS_SOLVER: Squared conjugate gradient • BICG_SOLVER: Biconjugate gradient • BICG_STAB_SOLVER: Biconjugate gradient stabilized • GMRES_SOLVER: Generalized Minimal Residual • QMR_SOLVER: Quasi-Minimal Residual Default value is CG_SOLVER
----	---------------	--

in	<i>prec</i>	Option to choose preconditioner in an enumerated variable <ul style="list-style-type: none"> • IDENT_PREC: Identity preconditioner (no preconditioning) • DIAG_PREC: Diagonal preconditioner [default] • ILU_PREC: ILU (Incomplete factorization) preconditioner Default value is DIAG_PREC
in	<i>max_it</i>	Maximum number of allowed iterations. Default value is 1000.
in	<i>toler</i>	Tolerance for convergence. Default value is 1.e-8

T_* get () const

Return C-Array.

Non zero terms of matrix is stored row by row.

T_ get (size_t i, size_t j) const [virtual]

Return entry (i,j) of matrix if this one is stored, 0 otherwise.

Parameters

in	<i>i</i>	Row index
in	<i>j</i>	Column index

Implements [Matrix< T_ >](#).

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return k-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_* a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

in	<i>el</i>	Pointer to element instance
in	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

<i>in</i>	<i>el</i>	Pointer to element instance
<i>in</i>	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

<i>in</i>	<i>sd</i>	Pointer to side instance
<i>in</i>	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

<i>in</i>	<i>sd</i>	Pointer to side instance
<i>in</i>	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
<i>in</i>	<i>code</i>	Code for which a boundary condition is to be enforced
<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe ([Vect](#)< T_ > & b, const [Vect](#)< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class [Mesh](#) & mesh, [Vect](#)< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe ([Vect](#)< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in</i>	<i>mesh</i>	Mesh instance from which information is extracted.
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

<i>in</i>	<i>dof</i>	Label of the concerned degree of freedom (DOF).
<i>in,out</i>	<i>b</i>	Vect instance that contains right-hand side.
<i>in</i>	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
<i>in</i>	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (<i>dof</i> >0) or both matrix and right-hand side (<i>dof</i> =0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side on input and solution on output
--------	----------	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- *n* if the *n*-th pivot is nul

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns *i*-th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	<i>i</i>	entry index
----	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns *k*-th stored element in matrix Index *k* starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.

Matrix< T_ > & operator+= (const T_ & x) [inherited]

Operator +=.

Add constant value x to all matrix entries.

Matrix< T_ > & operator-= (const Matrix< T_ > & m) [inherited]

Operator -=.

Subtract matrix m from current matrix instance.

Matrix< T_ > & operator-= (const T_ & x) [inherited]

Operator -=.

Subtract constant value x from all matrix entries.

Matrix< T_ > & operator*= (const T_ & x) [inherited]

Operator *.

Premultiply matrix entries by constant value x

void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax) [inherited]

Set a window for matrix printing.

Parameters

in	<i>rmin</i>	first row index to view
in	<i>rmax</i>	last row index to view
in	<i>cmin</i>	first column index to view
in	<i>cmax</i>	last column index to view

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const [inherited]

Return window data for matrix printing.

Parameters

out	<i>rmin</i>	first row index to view
out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.93 SteklovPoincare2DBE Class Reference

Solver of the Steklov Poincare problem in 2-D geometries using piecewise constant boundary elemen.

Public Member Functions

- [SteklovPoincare2DBE](#) (bool ext=false)
Default Constructor.
- [SteklovPoincare2DBE](#) (const [Mesh](#) &mesh, bool ext=false)
Constructor using mesh data.
- [SteklovPoincare2DBE](#) (const [Mesh](#) &mesh, const [Vect](#)< [real_t](#) > &g, [Vect](#)< [real_t](#) > &b, bool ext=false)

Constructor that solves the Steklov Poincare problem.

- `~SteklovPoincare2DBE ()`

Destructor.

- `void setMesh (const Mesh &mesh, bool ext=false)`

set Mesh instance

- `void Solve ()`

Build equation left and right-hand sides for P_0 (piecewise constant) approximation.

- `int Solve (Vect< real_t > &b, const Vect< real_t > &g)`

Build equation left and right-hand sides for P_0 (piecewise constant) approximation.

7.93.1 Detailed Description

Solver of the Steklov Poincare problem in 2-D geometries using piecewise constant boundary element.

`SteklovPoincare2DBE` solves the Steklov Poincare problem in 2-D: Given the trace of a harmonic function on the boundary of a given (inner or outer) domain, this class computes the normal derivative of the function. The normal is considered as oriented out of the bounded (inner) domain in both inner and outer configurations. The numerical approximation uses piecewise constant (P_0) approximation on edges of the boundary. Solution is obtained from the GMRES iterative solver without preconditioning. The given data is the vector (instance of class `Vect`) of piecewise constant values of the harmonic function on the boundary and the returned solution is piecewise constant value of the normal derivative considered either as a `Vect` instance.

Note

Although the mesh of the inner domain is not necessary to solve the problem, this one must be provided in order to calculate the outward normal.

7.93.2 Constructor & Destructor Documentation

`SteklovPoincare2DBE (bool ext = false)`

Default Constructor.

Parameters

<code>in</code>	<code>ext</code>	Boolean variable to say if the domain is external (true) or internal (false: Default value).
-----------------	------------------	--

`SteklovPoincare2DBE (const Mesh & mesh, bool ext = false)`

Constructor using mesh data.

This constructor calls member function `setMesh`.

Parameters

<code>in</code>	<code>mesh</code>	Reference to mesh instance.
<code>in</code>	<code>ext</code>	Boolean variable to say if the domain is external (true) or internal (false: Default value).

`SteklovPoincare2DBE (const Mesh & mesh, const Vect< real_t > & g, Vect< real_t > & b, bool ext = false)`

Constructor that solves the Steklov Poincare problem.

This constructor calls member function `setMesh` and `Solve`.

Parameters

in	<i>mesh</i>	Reference to mesh instance.
in	<i>g</i>	Vector (instance of class Vect) that contains imposed solution on the boundary
in	<i>b</i>	Vector (instance of class Vect) that contains the left hand side in input and the solution in output
in	<i>ext</i>	Boolean variable to say if the domain is external (true) or internal (false: Default value).

7.93.3 Member Function Documentation

void setMesh (const Mesh & mesh, bool ext = false)

set [Mesh](#) instance

Parameters

in	<i>mesh</i>	Mesh instance
in	<i>ext</i>	Boolean variable to say if the domain is external (true) or internal (false: Default value).

void Solve ()

Build equation left and right-hand sides for P_0 (piecewise constant) approximation.

This member function is to be used if the constructor using `mesh`, `b` and `g` has been used.

int Solve (Vect< real.t > & b, const Vect< real.t > & g)

Build equation left and right-hand sides for P_0 (piecewise constant) approximation.

This member function is to be used if the constructor using `mesh` has been used. It concerns cases where the imposed boundary condition is given by sides

Parameters

in	<i>g</i>	Vector that contains imposed solution on the boundary
in	<i>b</i>	Vector that contains the left hand side in input and the solution in output

7.94 Tabulation Class Reference

To read and manipulate tabulated functions.

Public Member Functions

- [Tabulation](#) ()
Default constructor.
- [Tabulation](#) (string file)
Constructor using file name.
- [~Tabulation](#) ()
Destructor.
- void [setFile](#) (string file)
Set file name.
- [real.t](#) [getValue](#) (string funct, [real.t](#) v)

Return the calculated value of the function.

- `real_t getDerivative` (string `funct`, `real_t v`)

Return the derivative of the function at a given point.

- `real_t getValue` (string `funct`, `real_t v1`, `real_t v2`)

Return the calculated value of the function.

- `real_t getValue` (string `funct`, `real_t v1`, `real_t v2`, `real_t v3`)

Return the calculated value of the function.

7.94.1 Detailed Description

To read and manipulate tabulated functions.

This class enables reading a tabulated function of one to three variables and calculating the value of the function using piecewise multilinear interpolation.

The file defining the function is an XML file where any function is introduced via the tag "Function".

7.94.2 Member Function Documentation

void setFile (string file)

Set file name.

This function is to be used when the default constructor is invoked.

real_t getValue (string funct, real_t v)

Return the calculated value of the function.

Case of a function of one variable

Parameters

<code>in</code>	<i>funct</i>	Name of the function to be evaluated, as read from input file
<code>in</code>	<i>v</i>	Value of the variable

Returns

Computed value of the function

real_t getDerivative (string funct, real_t v)

Return the derivative of the function at a given point.

Case of a function of one variable

Parameters

<code>in</code>	<i>funct</i>	Name of the function to be evaluated, as read from input file
<code>in</code>	<i>v</i>	Value of the variable

Returns

Derivative value

real_t getValue (string funct, real_t v1, real_t v2)

Return the calculated value of the function.

Case of a function of two variables

Parameters

in	<i>func</i>	Name of the function to be evaluated, as read from input file
in	<i>v1</i>	Value of the first variable
in	<i>v2</i>	Value of the second variable

Returns

Computed value of the function

real_t getValue (string func, real_t v1, real_t v2, real_t v3)

Return the calculated value of the function.

Case of a function of three variables

Parameters

in	<i>func</i>	Name of the funct to be evaluated, as read from input file
in	<i>v1</i>	Value of the first variable
in	<i>v2</i>	Value of the second variable
in	<i>v3</i>	Value of the third variable

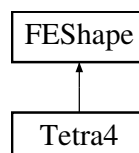
Returns

Computed value of the function

7.95 Tetra4 Class Reference

Defines a three-dimensional 4-node tetrahedral finite element using P_1 interpolation.

Inheritance diagram for Tetra4:



Public Member Functions

- **Tetra4** ()
Default Constructor.
- **Tetra4** (const **Element** *el)
*Constructor when data of **Element** *el* are given.*
- **~Tetra4** ()
Destructor.
- void **set** (const **Element** *el)
Choose element by giving its pointer.
- **real_t Sh** (size_t i, **Point**< **real_t** > s) const
*Calculate shape function of node *i* at a given point *s*.*
- **Point**< **real_t** > **DSh** (size_t i) const
*Return *x*, *y* and *z* partial derivatives of shape function associated to node *i*.*
- **real_t getVolume** () const

- Return volume of element.*
- `Point< real_t > getRefCoord (const Point< real_t > &x) const`
Return reference coordinates of a point x in element.
- `bool isIn (const Point< real_t > &x)`
Check whether point x is in current tetrahedron or not.
- `real_t getInterpolate (const Point< real_t > &x, const LocalVect< real_t, 4 > &v)`
Return interpolated value at point of coordinate x
- `Point< real_t > EdgeSh (size_t k, Point< real_t > s)`
Return edge shape function.
- `Point< real_t > CurlEdgeSh (size_t k)`
Return curl of edge shape function.
- `real_t getMaxEdgeLength () const`
Return maximal edge length of tetrahedron.
- `real_t getMinEdgeLength () const`
Return minimal edge length of tetrahedron.
- `real_t Sh (size_t i) const`
Return shape function of node i at given point.
- `real_t getDet () const`
Return determinant of jacobian.
- `Point< real_t > getCenter () const`
Return coordinates of center of element.
- `Point< real_t > getLocalPoint () const`
Localize a point in the element.
- `Point< real_t > getLocalPoint (const Point< real_t > &s) const`
Localize a point in the element.

7.95.1 Detailed Description

Defines a three-dimensional 4-node tetrahedral finite element using P_1 interpolation.

The reference element is the right tetrahedron with four unit edges interpolation.

7.95.2 Member Function Documentation

`real_t Sh (size_t i, Point< real_t > s) const`

Calculate shape function of node i at a given point s .

s is a point in the reference tetrahedron.

`Point<real_t> DSh (size_t i) const`

Return x , y and z partial derivatives of shape function associated to node i .

Note that these are constant in element.

`Point<real_t> EdgeSh (size_t k, Point< real_t > s)`

Return edge shape function.

Parameters

<code>in</code>	<code>k</code>	Local edge number for which the edge shape function is computed
<code>in</code>	<code>s</code>	Local coordinates in element

Remarks

`Element` edges are ordered as follows: `Edge k` has end vertices `k` and `k+1`

Point<real_t> CurlEdgeSh (size_t k)

Return curl of edge shape function.

Parameters

<code>in</code>	<code>k</code>	Local edge number for which the curl of the edge shape function is computed
-----------------	----------------	---

Remarks

`Element` edges are ordered as follows: `Edge k` has end vertices `k` and `k+1`

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

Localize a point in the element.

Return actual coordinates where `s` are coordinates in the reference element.

7.96 Timer Class Reference

To handle elapsed time counting.

Public Member Functions

- `Timer ()`
Default constructor.
- `~Timer ()`
Destructor.
- `bool Started () const`
Say if time counter has started.
- `void Start ()`
Start time counting.

- void `Stop` ()
Stop time counting.
- void `Resume` ()
Resume counting.
- void `Clear` ()
Clear time value (Set to zero)
- double `get` () const
Return elapsed time (in seconds)
- `real_t` `getTime` () const
Return elapsed time (in seconds)

7.96.1 Detailed Description

To handle elapsed time counting.

This class is to be used when testing program performances. A normal usage of the class is, once an instance is constructed, to use alternatively, Start, Stop and Resume. Elapsed time can be obtained once the member function Stop is called.

7.96.2 Member Function Documentation

bool Started () const

Say if time counter has started.

Return true if time has started, false if not

void Start ()

Start time counting.

This member function is to be used at first or if the user wants to clear time counting

void Stop ()

Stop time counting.

This function interrupts time counting. This one can be resumed by the function Resume

real_t getTime () const

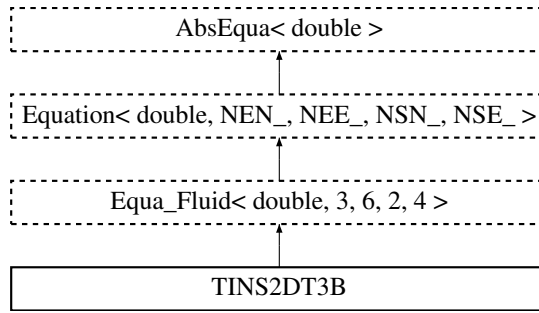
Return elapsed time (in seconds)

Identical to get

7.97 TINS2DT3B Class Reference

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Inheritance diagram for TINS2DT3B:



Public Member Functions

- **TINS2DT3B** ()
Default Constructor.
- **TINS2DT3B** (Mesh &mesh, Vect< real.t > &u, Vect< real.t > &p, real.t &ts, real.t Re=0.)
Constructor using mesh.
- **~TINS2DT3B** ()
Destructor.
- void **setInput** (EqDataType opt, Vect< real.t > &u)
Set equation input data.
- int **runOneTimeStep** ()
Run one time step.
- int **run** ()
Run (in the case of one step run)
- void **updateBC** (const Element &el, const Vect< double > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **updateBC** (const Vect< double > &bc)
Update Right-Hand side by taking into account essential boundary conditions.
- void **DiagBC** (int dof_type=NODE_DOF, int dof=0)
Update element matrix to impose bc by diagonalization technique.
- void **LocalNodeVector** (Vect< double > &b)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const Vect< double > &b, LocalVect< double, NEE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementNodeVector** (const Vect< double > &b, LocalVect< double, NEN_ > &be, int dof)
Localize Element Vector from a Vect instance.
- void **ElementNodeVectorSingleDOF** (const Vect< double > &b, LocalVect< double, NEN_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementSideVector** (const Vect< double > &b, LocalVect< double, NSE_ > &be)
Localize Element Vector from a Vect instance.
- void **ElementVector** (const Vect< double > &b, int dof_type=NODE_FIELD, int flag=0)
Localize Element Vector.
- void **SideVector** (const Vect< double > &b)
Localize Side Vector.
- void **ElementNodeCoordinates** ()
Localize coordinates of element nodes.

- void [SideNodeCoordinates](#) ()
Localize coordinates of side nodes.
- void [ElementAssembly](#) ([Matrix](#)< double > *A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkSMatrix](#)< double > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SkMatrix](#)< double > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([SpMatrix](#)< double > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([TrMatrix](#)< double > &A)
Assemble element matrix into global one.
- void [ElementAssembly](#) ([Vect](#)< double > &v)
Assemble element vector into global one.
- void [DGElementAssembly](#) ([Matrix](#)< double > *A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkSMatrix](#)< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SkMatrix](#)< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([SpMatrix](#)< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [DGElementAssembly](#) ([TrMatrix](#)< double > &A)
Assemble element matrix into global one for the Discontinuous Galerkin approximation.
- void [SideAssembly](#) ([Matrix](#)< double > *A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SkSMatrix](#)< double > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SkMatrix](#)< double > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([SpMatrix](#)< double > &A)
Assemble side (edge or face) matrix into global one.
- void [SideAssembly](#) ([Vect](#)< double > &v)
Assemble side (edge or face) vector into global one.
- void [AxbAssembly](#) (const [Element](#) &el, const [Vect](#)< double > &x, [Vect](#)< double > &b)
Assemble product of element matrix by element vector into global vector.
- void [AxbAssembly](#) (const [Side](#) &sd, const [Vect](#)< double > &x, [Vect](#)< double > &b)
Assemble product of side matrix by side vector into global vector.
- size_t [getNbNodes](#) () const
Return number of element nodes.
- size_t [getNbEq](#) () const
Return number of element equations.
- double * [A](#) ()
Return element matrix as a C-array.
- double * [sA](#) ()
Return side matrix as a C-array.

- `double * b ()`
Return element right-hand side as a C-array.
- `double * sb ()`
Return side right-hand side as a C-array.
- `double * Prev ()`
Return element matrix as a C-array.
- `LocalMatrix< double, NEE_, NEE_ > & EA ()`
Return element matrix as a LocalMatrix instance.
- `LocalMatrix< double, NSE_, NSE_ > & SA ()`
Return side matrix as a LocalMatrix instance.
- `LocalVect< double, NEE_ > & Eb ()`
Return element right-hand side as a LocalVect instance.
- `LocalVect< double, NEE_ > & Ep ()`
Return element matrix as a C-array.
- `void setInitialSolution (const Vect< double > &u)`
Set initial solution (previous time step)
- `real_t setMaterialProperty (const string &exp, const string &prop)`
Define a material property by an algebraic expression.
- `void setMesh (class Mesh &m)`
Define mesh and renumber DOFs after removing imposed ones.
- `Mesh & getMesh () const`
Return reference to Mesh instance.
- `LinearSolver< double > & getLinearSolver ()`
Return reference to linear solver instance.
- `void setSolver (int ls, int pc=IDENT_PREC)`
Choose solver for the linear system.
- `int solveEigenProblem (int nb_eigv, bool g=false)`
Compute eigenvalues and eigenvectors.
- `real_t getEigenValue (int n) const`
Return the n-th eigenvalue.
- `void getEigenvector (int n, Vect< real_t > &v) const`
Store the eigenvector corresponding to a given eigenvalue.
- `class Eigen & getEigenSolver ()`
Return reference to eigenproblem solver.

Protected Member Functions

- `void Viscosity (const real_t &visc)`
Set (constant) Viscosity.
- `void Viscosity (const string &exp)`
Set viscosity given by an algebraic expression.
- `void Density (const real_t &dens)`
Set (constant) Viscosity.
- `void Density (const string &exp)`
Set Density given by an algebraic expression.
- `void ThermalExpansion (const real_t *e)`
Set (constant) thermal expansion coefficient.

- void [ThermalExpansion](#) (const string &exp)
Set thermal expansion coefficient given by an algebraic expression.
- void [setMaterial](#) ()
Set material properties.
- void [Init](#) (const [Element](#) *el)
Set element arrays to zero.
- void [Init](#) (const [Side](#) *sd)
Set side arrays to zero.

7.97.1 Detailed Description

Builds finite element arrays for thermal diffusion and convection in 2-D domains using 3-Node triangles.

Note that members calculating element arrays have as an argument a double coef that will be multiplied by the contribution of the current element. This makes possible testing different algorithms.

7.97.2 Constructor & Destructor Documentation

TINS2DT3B ([Mesh](#) & mesh, [Vect](#)< real_t > & u, [Vect](#)< real_t > & p, real_t & ts, real_t Re = 0.)

Constructor using mesh.

Parameters

in	<i>mesh</i>	Mesh instance
in,out	<i>u</i>	Vect instance containing initial velocity. This vector is updated during computations and will therefore contain velocity at each time step
out	<i>p</i>	Vect instance that will contain pressure at nodes. This vector is updated during computations and will therefore contain pressure at each time step
in	<i>ts</i>	Time step
in	<i>Re</i>	Reynolds number. The default value (0) means that no Reynolds number is given and problem data are supplied by material properties. If Re has any other value, then nondimensional form of the equations is assumed and material properties are ignored.

7.97.3 Member Function Documentation

void setInput ([EqDataType](#) opt, [Vect](#)< real_t > & u)

Set equation input data.

Parameters

in	<i>opt</i>	Parameter to select type of input (enumerated values) <ul style="list-style-type: none"> • INITIAL_FIELD: Initial temperature • BOUNDARY_CONDITION_DATA: Boundary condition (Dirichlet) • SOURCE_DATA: Heat source • FLUX_DATA: Heat flux (Neumann boundary condition) • VELOCITY_FIELD: Velocity vector (for the convection term)
in	<i>u</i>	Vector containing input data (Vect instance)

void updateBC (const Element & el, const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>el</i>	Reference to current element instance
in	<i>bc</i>	Vector that contains imposed values at all DOFs

void updateBC (const Vect< double > & bc) [inherited]

Update Right-Hand side by taking into account essential boundary conditions.

Parameters

in	<i>bc</i>	Vector that contains imposed values at all DOFs
-----------	-----------	---

Remarks

The current element is pointed by `_theElement`

void DiagBC (int dof_type = *NODE_DOF*, int dof = 0) [inherited]

Update element matrix to impose bc by diagonalization technique.

Parameters

in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>dof</i>	DOF setting: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF No. <code>dof</code> is handled in the system

void LocalNodeVector (Vect< double > & b) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Reference to global vector to be localized. The resulting local vector can be accessed by attribute <code>ePrev</code> . This member function is to be used if a constructor with <code>Element</code> was invoked.
-----------	----------	---

void ElementNodeVector (const Vect< double > & b, LocalVect< double , NEE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

All degrees of freedom are transferred to the local vector

void ElementNodeVector (const Vect< double > & b, LocalVect< double , NEN_ > & be, int dof) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.
in	<i>dof</i>	Degree of freedom to transfer to the local vector

Remarks

Only yhe dega dof is transferred to the local vector

void ElementNodeVectorSingleDOF (const Vect< double > & b, LocalVect< double , NEN_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is the total number of element equations.

Remarks

Vector b is assumed to contain only one degree of freedom by node.

void ElementSideVector (const Vect< double > & b, LocalVect< double , NSE_ > & be) [inherited]

Localize Element Vector from a Vect instance.

Parameters

in	<i>b</i>	Global vector to be localized.
out	<i>be</i>	Local vector, the length of which is

void ElementVector (const Vect< double > & b, int dof_type = NODE_FIELD, int flag = 0) [inherited]

Localize Element Vector.

Parameters

in	<i>b</i>	Global vector to be localized
in	<i>dof_type</i>	DOF type option. To choose among the enumerated values: <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [Default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides
in	<i>flag</i>	Option to set: <ul style="list-style-type: none"> • <code>= 0</code>, All DOFs are taken into account [Default] • <code>!= 0</code>, Only DOF number <code>dof</code> is handled in the system The resulting local vector can be accessed by attribute <code>ePrev</code> .

Remarks

This member function is to be used if a constructor with `Element` was invoked. It uses the `Element` pointer `_theElement`

void SideVector (const Vect< double > & b) [inherited]

Localize Side Vector.

Parameters

in	<i>b</i>	Global vector to be localized <ul style="list-style-type: none"> • <code>NODE.FIELD</code>, DOFs are supported by nodes [default] • <code>ELEMENT.FIELD</code>, DOFs are supported by elements • <code>SIDE.FIELD</code>, DOFs are supported by sides The resulting local vector can be accessed by attribute <code>ePrev</code> .
-----------	----------	---

Remarks

This member function is to be used if a constructor with `Side` was invoked. It uses the `Side` pointer `_theSide`

void ElementNodeCoordinates () [inherited]

Localize coordinates of element nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the `Side` pointer `_theSide`

void SideNodeCoordinates () [inherited]

Localize coordinates of side nodes.

Coordinates are stored in array `_x[0]`, `_x[1]`, ... which are instances of class [Point<real.t>](#)

Remarks

This member function uses the Element pointer `_theElement`

void ElementAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

A	Global matrix stored as an SkSMatrix instance
----------	---

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void ElementAssembly (Vect< double > & v) [inherited]

Assemble element vector into global one.

Parameters

in	v	Global vector (Vect instance)
----	---	-------------------------------

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (Matrix< double > * A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkSMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

A	Global matrix stored as an SkSMatrix instance
---	---

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SkMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (SpMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an SpMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void DGElementAssembly (TrMatrix< double > & A) [inherited]

Assemble element matrix into global one for the Discontinuous Galerkin approximation.

Parameters

in	A	Global matrix stored as an TrMatrix instance
----	---	--

Warning

The element pointer is given by the global variable `theElement`

void SideAssembly (Matrix< double > * A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

A	Pointer to global matrix (abstract class: can be any of classes SkSMatrix, SkMatrix, SpMatrix)
---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkSMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkSMatrix instance
----	---	---

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SkMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	A	Global matrix stored as an SkMatrix instance
----	---	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (SpMatrix< double > & A) [inherited]

Assemble side (edge or face) matrix into global one.

Parameters

in	<i>A</i>	Global matrix stored as an SpMatrix instance
-----------	----------	--

Warning

The side pointer is given by the global variable `theSide`

void SideAssembly (Vect< double > & v) [inherited]

Assemble side (edge or face) vector into global one.

Parameters

in	<i>v</i>	Global vector (Vect instance)
-----------	----------	-------------------------------

Warning

The side pointer is given by the global variable `theSide`

void AxbAssembly (const Element & el, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of element matrix by element vector into global vector.

Parameters

in	<i>el</i>	Reference to Element instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector to add (Vect instance)

void AxbAssembly (const Side & sd, const Vect< double > & x, Vect< double > & b) [inherited]

Assemble product of side matrix by side vector into global vector.

Parameters

in	<i>sd</i>	Reference to Side instance
in	<i>x</i>	Global vector to multiply by (Vect instance)
out	<i>b</i>	Global vector (Vect instance)

real.t setMaterialProperty (const string & exp, const string & prop) [inherited]

Define a material property by an algebraic expression.

Parameters

in	<i>exp</i>	Algebraic expression
in	<i>prop</i>	Property name

Returns

Return value in expression evaluation:

- =0, Normal evaluation
- !=0, An error message is displayed

Mesh& getMesh () const [inherited]

Return reference to Mesh instance.

Returns

Reference to Mesh instance

void setSolver (int ls, int pc = IDENT_PREC) [inherited]

Choose solver for the linear system.

Parameters

in	ls	<p>Solver of the linear system. To choose among the enumerated values: DIRECT_SOLVER, CG_SOLVER, GMRES_SOLVER</p> <ul style="list-style-type: none"> • DIRECT_SOLVER, Use a facorization solver [default] • CG_SOLVER, Conjugate Gradient iterative solver • CGS_SOLVER, Squared Conjugate Gradient iterative solver • BICG_SOLVER, BiConjugate Gradient iterative solver • BICG_STAB_SOLVER, BiConjugate Gradient Stabilized iterative solver • GMRES_SOLVER, GMRES iterative solver • QMR_SOLVER, QMR iterative solver
in	pc	<p>Preconditioner to associate to the iterative solver. If the direct solver was chosen for the first argument this argument is not used. Otherwise choose among the enumerated values:</p> <ul style="list-style-type: none"> • IDENT_PREC, Identity preconditioner (no preconditioning [default]) • DIAG_PREC, Diagonal preconditioner • ILU_PREC, Incomplete LU factorization preconditioner

int solveEigenProblem (int nb_eigv, bool g = false) [inherited]

Compute eigenvalues and eigenvectors.

Eigenvalues and vectors are computed using the Bathe's subspace iteration method.

Parameters

in	nb_eigv	Number of eigenvalues to compute
in	g	Option to choose whether to solve a generalized eigenvalue problem (true) or a standard one (false). The generalized eigenvalue problem corresponds to the case where a consistent mass matrix (rather than a lumped one) is computed. Default value is false.

real.t getEigenValue (int n) const [inherited]

Return the n-th eigenvalue.

This functions works only if the member function `getEigen` was called with an argument `nb_eigv` greater or equal to `n`. Otherwise it returns 0.

void getEigenVector (int n, Vect< real_t > & v) const [inherited]

Store the eigenvector corresponding to a given eigenvalue.

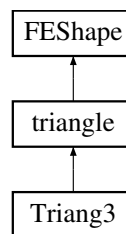
Parameters

in	n	Label of the eigenvalue
out	v	Vect instance containing the corresponding eigenvector. This vector is resized.

7.98 Triang3 Class Reference

Defines a 3-Node (P_1) triangle.

Inheritance diagram for Triang3:



Public Member Functions

- [Triang3](#) ()
Default Constructor.
- [Triang3](#) (const [Element](#) *el)
Constructor for an element.
- [Triang3](#) (const [Side](#) *sd)
Constructor for a side.
- [~Triang3](#) ()
Destructor.
- void [set](#) (const [Element](#) *el)
Choose element by giving its pointer.
- void [set](#) (const [Side](#) *sd)
Choose side by giving its pointer.
- [real_t](#) [Sh](#) (size_t i, [Point](#)< [real_t](#) > s) const
Calculate shape function of node at a given point.
- [Point](#)< [real_t](#) > [DSh](#) (size_t i) const
Calculate derivatives of shape function of node i
- [real_t](#) [getInterpolate](#) (const [Point](#)< [real_t](#) > &x, const [LocalVect](#)< [real_t](#), 3 > &v)
Return interpolated value at point of coordinate x
- [real_t](#) [check](#) () const
Check element area and number of nodes.
- [Point](#)< [real_t](#) > [Grad](#) (const [LocalVect](#)< [real_t](#), 3 > &u) const

- Return constant gradient vector in triangle.*
 - `real_t getMaxEdgeLength () const`
Return maximal edge length of triangle.
 - `real_t getMinEdgeLength () const`
Return minimal edge length of triangle.
 - `real_t getArea ()`
Return element area.
 - `Point< real_t > getCenter () const`
Return coordinates of center of element.
 - `Point< real_t > getCircumcenter () const`
Return coordinates of circumcenter of element.
 - `real_t getCircumRadius () const`
Return radius of circumscribed circle of triangle.
 - `real_t getInRadius () const`
Return radius of inscribed circle of triangle.
 - `Point< real_t > getRefCoord (const Point< real_t > &x) const`
Return reference coordinates of a point x in element.
 - `bool isIn (const Point< real_t > &x) const`
Check whether point x is in current triangle or not.
 - `bool isStrictlyIn (const Point< real_t > &x) const`
Check whether point x is strictly in current triangle (not on the boundary) or not.
 - `real_t Sh (size_t i) const`
Return shape function of node i at given point.
 - `real_t getDet () const`
Return determinant of jacobian.
 - `Point< real_t > getLocalPoint () const`
Localize a point in the element.
 - `Point< real_t > getLocalPoint (const Point< real_t > &s) const`
Localize a point in the element.

7.98.1 Detailed Description

Defines a 3-Node (P_1) triangle.

The reference element is the rectangle triangle with two unit edges.

7.98.2 Constructor & Destructor Documentation

Triang3 (const Element * el)

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

Triang3 (const Side * sd)

Constructor for a side.

The constructed triangle is a side in a 3-D mesh.

7.98.3 Member Function Documentation

real_t Sh (size_t i, Point< real_t > s) const

Calculate shape function of node at a given point.

Parameters

in	<i>i</i>	Label (local) of node
in	<i>s</i>	Natural coordinates of node where to evaluate

real_t check () const

Check element area and number of nodes.

Returns

- > 0 : m is the length
- $= 0$: zero length (\Rightarrow Error)

Point<real_t> Grad (const LocalVect< real_t, 3 > & u) const

Return constant gradient vector in triangle.

Parameters

in	<i>u</i>	Local vector for which the gradient is evaluated
-----------	----------	--

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element \rightarrow Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element \rightarrow Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

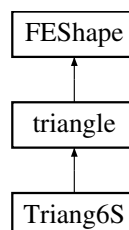
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.99 Triang6S Class Reference

Defines a 6-Node straight triangular finite element using P_2 interpolation.

Inheritance diagram for Triang6S:



Public Member Functions

- [Triang6S](#) ()
Default Constructor.
- [Triang6S](#) (const [Element](#) *el)
Constructor for an element.
- [~Triang6S](#) ()
Destructor.
- [real_t](#) Sh (size_t i, const [Point](#)< [real_t](#) > &s) const
Calculate shape function of a node.
- [Point](#)< [real_t](#) > DSh (size_t i, const [Point](#)< [real_t](#) > &s) const
Calculate derivatives of shape function of a node.
- [Point](#)< [real_t](#) > [getCenter](#) () const
Return coordinates of center of element.
- [Point](#)< [real_t](#) > [Grad](#) (const [LocalVect](#)< [real_t](#), 6 > &u, const [Point](#)< [real_t](#) > &s) const
Return gradient vector in triangle at a given point.
- [real_t](#) [getMaxEdgeLength](#) () const
Return maximal edge length of triangle.
- [real_t](#) [getMinEdgeLength](#) () const
Return minimal edge length of triangle.
- [real_t](#) [getArea](#) ()
Return element area.
- [Point](#)< [real_t](#) > [getCircumcenter](#) () const
Return coordinates of circumcenter of element.
- [real_t](#) [getCircumRadius](#) () const
Return radius of circumscribed circle of triangle.
- [real_t](#) [getInRadius](#) () const
Return radius of inscribed circle of triangle.
- [Point](#)< [real_t](#) > [getRefCoord](#) (const [Point](#)< [real_t](#) > &x) const
Return reference coordinates of a point x in element.
- bool [isIn](#) (const [Point](#)< [real_t](#) > &x) const
Check whether point x is in current triangle or not.
- bool [isStrictlyIn](#) (const [Point](#)< [real_t](#) > &x) const
Check whether point x is strictly in current triangle (not on the boundary) or not.
- [real_t](#) Sh (size_t i) const
Return shape function of node i at given point.
- [real_t](#) Sh (size_t i, [Point](#)< [real_t](#) > s) const
Calculate shape function of node i at a given point s.
- [Point](#)< [real_t](#) > DSh (size_t i) const
Return derivatives of shape function of node i at a given point.
- [real_t](#) [getDet](#) () const
Return determinant of jacobian.
- [Point](#)< [real_t](#) > [getLocalPoint](#) () const
Localize a point in the element.
- [Point](#)< [real_t](#) > [getLocalPoint](#) (const [Point](#)< [real_t](#) > &s) const
Localize a point in the element.

7.99.1 Detailed Description

Defines a 6-Node straight triangular finite element using P_2 interpolation.

The reference element is the rectangle triangle with two unit edges.

7.99.2 Constructor & Destructor Documentation

Triang6S (const Element * el)

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

Parameters

in	el	Pointer to Element instance
----	----	---

7.99.3 Member Function Documentation

real_t Sh (size_t i, const Point< real_t > & s) const

Calculate shape function of a node.

Parameters

in	i	Local label of the node $1 \leq i \leq 6$
in	s	Local coordinates of the point where the shape function is evaluated

Point<real_t> DSh (size_t i, const Point< real_t > & s) const

Calculate derivatives of shape function of a node.

Parameters

in	i	Local label of node
in	s	Local coordinates of the point where the gradient of the shape function is evaluated

Point<real_t> Grad (const LocalVect< real_t, 6 > & u, const Point< real_t > & s) const

Return gradient vector in triangle at a given point.

Parameters

in	s	Local coordinates of the point where the gradient of the shape function is evaluated
in	u	Local vector for which the gradient is evaluated

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

in	i	Local node label
in	s	Point in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const [inherited]

Return derivatives of shape function of node *i* at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function `setLocal()` must have been called before in order to calculate relevant quantities.

Parameters

<code>in</code>	<code>i</code>	Partial derivative index (1, 2 or 3)
-----------------	----------------	--------------------------------------

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

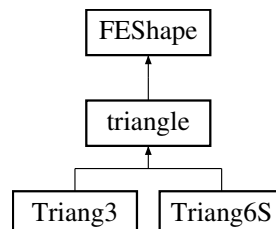
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.100 triangle Class Reference

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

Inheritance diagram for triangle:



Public Member Functions

- **triangle ()**
Default Constructor.
- **triangle (const Element *el)**
Constructor for an element.
- **triangle (const Side *sd)**
Constructor for a side.
- **virtual ~triangle ()**
Destructor.
- **real_t getArea ()**
Return element area.
- **Point< real_t > getCenter () const**
Return coordinates of center of element.
- **Point< real_t > getCircumcenter () const**

- Return coordinates of circumcenter of element.*
 - `real_t getCircumRadius () const`
- Return radius of circumscribed circle of triangle.*
 - `real_t getInRadius () const`
- Return radius of inscribed circle of triangle.*
 - `Point< real_t > getRefCoord (const Point< real_t > &x) const`
- Return reference coordinates of a point x in element.*
 - `real_t getMaxEdgeLength () const`
- Return maximal edge length of triangle.*
 - `real_t getMinEdgeLength () const`
- Return minimal edge length of triangle.*
 - `bool isIn (const Point< real_t > &x) const`
- Check whether point x is in current triangle or not.*
 - `bool isStrictlyIn (const Point< real_t > &x) const`
- Check whether point x is strictly in current triangle (not on the boundary) or not.*
 - `real_t Sh (size_t i) const`
- Return shape function of node i at given point.*
 - `real_t Sh (size_t i, Point< real_t > s) const`
- Calculate shape function of node i at a given point s.*
 - `Point< real_t > DSh (size_t i) const`
- Return derivatives of shape function of node i at a given point.*
 - `real_t getDet () const`
- Return determinant of jacobian.*
 - `Point< real_t > getLocalPoint () const`
- Localize a point in the element.*
 - `Point< real_t > getLocalPoint (const Point< real_t > &s) const`
- Localize a point in the element.*

7.100.1 Detailed Description

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

7.100.2 Constructor & Destructor Documentation

triangle (const Element * el)

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

triangle (const Side * sd)

Constructor for a side.

The constructed triangle is a side in a 3-D mesh.

7.100.3 Member Function Documentation

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

in	<i>i</i>	Local node label
in	<i>s</i>	Point in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const [inherited]

Return derivatives of shape function of node *i* at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Parameters

in	<i>i</i>	Partial derivative index (1, 2 or 3)
----	----------	--------------------------------------

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

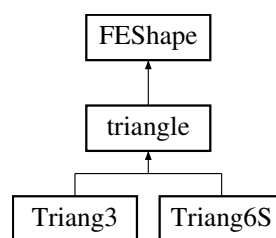
Localize a point in the element.

Return actual coordinates where *s* are coordinates in the reference element.

7.101 triangle Class Reference

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

Inheritance diagram for triangle:



Public Member Functions

- [triangle](#) ()
Default Constructor.
- [triangle](#) (const [Element](#) *el)
Constructor for an element.
- [triangle](#) (const [Side](#) *sd)

Constructor for a side.

- `virtual ~triangle ()`

Destructor.

- `real_t getArea ()`

Return element area.

- `Point< real_t > getCenter () const`

Return coordinates of center of element.

- `Point< real_t > getCircumcenter () const`

Return coordinates of circumcenter of element.

- `real_t getCircumRadius () const`

Return radius of circumscribed circle of triangle.

- `real_t getInRadius () const`

Return radius of inscribed circle of triangle.

- `Point< real_t > getRefCoord (const Point< real_t > &x) const`

Return reference coordinates of a point x in element.

- `real_t getMaxEdgeLength () const`

Return maximal edge length of triangle.

- `real_t getMinEdgeLength () const`

Return minimal edge length of triangle.

- `bool isIn (const Point< real_t > &x) const`

Check whether point x is in current triangle or not.

- `bool isStrictlyIn (const Point< real_t > &x) const`

Check whether point x is strictly in current triangle (not on the boundary) or not.

- `real_t Sh (size_t i) const`

Return shape function of node i at given point.

- `real_t Sh (size_t i, Point< real_t > s) const`

Calculate shape function of node i at a given point s .

- `Point< real_t > DSh (size_t i) const`

Return derivatives of shape function of node i at a given point.

- `real_t getDet () const`

Return determinant of jacobian.

- `Point< real_t > getLocalPoint () const`

Localize a point in the element.

- `Point< real_t > getLocalPoint (const Point< real_t > &s) const`

Localize a point in the element.

7.101.1 Detailed Description

Defines a triangle. The reference element is the rectangle triangle with two unit edges.

7.101.2 Constructor & Destructor Documentation

`triangle (const Element * el)`

Constructor for an element.

The constructed triangle is an element in a 2-D mesh.

triangle (const Side * sd)

Constructor for a side.

The constructed triangle is a side in a 3-D mesh.

7.101.3 Member Function Documentation

real_t Sh (size_t i, Point< real_t > s) const [inherited]

Calculate shape function of node i at a given point s.

Parameters

in	i	Local node label
in	s	Point in the reference triangle where the shape function is evaluated

Point<real_t> DSh (size_t i) const [inherited]

Return derivatives of shape function of node i at a given point.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Parameters

in	i	Partial derivative index (1, 2 or 3)
----	---	--------------------------------------

real_t getDet () const [inherited]

Return determinant of jacobian.

If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint () const [inherited]

Localize a point in the element.

Return actual coordinates in the reference element. If the transformation (Reference element -> Actual element) is not affine, member function **setLocal()** must have been called before in order to calculate relevant quantities.

Point<real_t> getLocalPoint (const Point< real_t > & s) const [inherited]

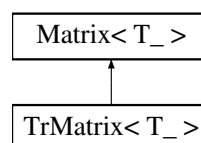
Localize a point in the element.

Return actual coordinates where s are coordinates in the reference element.

7.102 TrMatrix< T_ > Class Template Reference

To handle tridiagonal matrices.

Inheritance diagram for TrMatrix< T_ >:



Public Member Functions

- [TrMatrix](#) ()
Default constructor.
- [TrMatrix](#) (size_t size)
Constructor for a tridiagonal matrix with size rows.
- [TrMatrix](#) (const [TrMatrix](#) &m)
Copy Constructor.
- [~TrMatrix](#) ()

Destructor.

- void `setSize` (size_t `size`)
Set size (number of rows) of matrix.
- void `MultAdd` (const `Vect`< T_ > &`x`, `Vect`< T_ > &`y`) const
Multiply matrix by vector x and add result to y .
- void `MultAdd` (T_ `a`, const `Vect`< T_ > &`x`, `Vect`< T_ > &`y`) const
*Multiply matrix by vector $a*x$ and add result to y .*
- void `Mult` (const `Vect`< T_ > &`x`, `Vect`< T_ > &`y`) const
Multiply matrix by vector x and save result in y .
- void `TMult` (const `Vect`< T_ > &`x`, `Vect`< T_ > &`y`) const
Multiply transpose of matrix by vector x and save result in y .
- void `Axpy` (T_ `a`, const `TrMatrix`< T_ > &`m`)
Add to matrix the product of a matrix by a scalar.
- void `Axpy` (T_ `a`, const `Matrix`< T_ > *`m`)
Add to matrix the product of a matrix by a scalar.
- void `set` (size_t `i`, size_t `j`, const T_ &`val`)
Assign constant val to an entry (i, j) of the matrix.
- void `add` (size_t `i`, size_t `j`, const T_ &`val`)
Add constant val value to an entry (i, j) of the matrix.
- T_ `operator()` (size_t `i`, size_t `j`) const
Operator $()$ (Constant version).
- T_ & `operator()` (size_t `i`, size_t `j`)
Operator $()$ (Non constant version).
- `TrMatrix`< T_ > & `operator=` (const `TrMatrix`< T_ > &`m`)
Operator $=$.
- `TrMatrix`< T_ > & `operator=` (const T_ &`x`)
Operator $=$.
- `TrMatrix`< T_ > & `operator*=` (const T_ &`x`)
Operator $=$.*
- int `Solve` (`Vect`< T_ > &`b`)
Solve a linear system with current matrix (forward and back substitution).
- T_ * `get` () const
Return C-Array.
- T_ `get` (size_t `i`, size_t `j`) const
Return entry (i, j) of matrix.
- size_t `getNbRows` () const
Return number of rows.
- size_t `getNbColumns` () const
Return number of columns.
- void `setPenal` (real_t `p`)
Set Penalty Parameter (For boundary condition prescription).
- void `setDiagonal` ()
Set the matrix as diagonal.
- void `setDiagonal` (const class `Mesh` &`mesh`)
Initialize matrix storage in the case where only diagonal terms are stored.
- T_ `getDiag` (size_t `k`) const

- Return k-th diagonal entry of matrix.*

 - `size_t size () const`
- Return matrix dimension (Number of rows and columns).*

 - `void Assembly (const class Element *el, T_ *a)`
- Assembly of element matrix into global matrix.*

 - `void Assembly (const class Element *el, const DMatrix< T_ > &a)`
- Assembly of element matrix into global matrix.*

 - `void Assembly (const class Side *sd, T_ *a)`
- Assembly of side matrix into global matrix.*

 - `void Assembly (const class Side *sd, const DMatrix< T_ > &a)`
- Assembly of side matrix into global matrix.*

 - `void Prescribe (const class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
- Impose by a penalty method an essential boundary condition.*

 - `void Prescribe (int dof, int code, const class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
- Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.*

 - `void Prescribe (Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
- Impose by a penalty method an essential boundary condition, using the Mesh instance provided by the constructor.*

 - `void Prescribe (const class Mesh &mesh, Vect< T_ > &b, int flag=0)`
- Impose by a penalty method a homogeneous (=0) essential boundary condition.*

 - `void Prescribe (Vect< T_ > &b, int flag=0)`
- Impose by a penalty method a homogeneous (=0) essential boundary condition, using the Mesh instance provided by the constructor.*

 - `void Prescribe (size_t dof, const class Mesh &mesh, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
- Impose by a penalty method an essential boundary condition when only one DOF is treated.*

 - `void Prescribe (size_t dof, Vect< T_ > &b, const Vect< T_ > &u, int flag=0)`
- Impose by a penalty method an essential boundary condition when only one DOF is treated.*

 - `void PrescribeSide (const class Mesh &mesh)`
- Impose by a penalty method an essential boundary condition when DOFs are supported by sides.*

 - `void PrescribeSide ()`
- Impose by a penalty method an essential boundary condition when DOFs are supported by sides.*

 - `virtual int Factor ()=0`
- Factorize matrix. Available only if the storage class enables it.*

 - `int Solve (const Vect< T_ > &b, Vect< T_ > &x)`
- Solve system with factorized matrix (forward and back substitution).*

 - `int FactorAndSolve (Vect< T_ > &b)`
- Factorize matrix and solve the linear system.*

 - `int FactorAndSolve (const Vect< T_ > &b, Vect< T_ > &x)`
- Factorize matrix and solve the linear system.*

 - `unsigned long getLength () const`
- Return number of stored terms in matrix.*

 - `int isDiagonal () const`
- Say if matrix is diagonal or not.*

 - `int isFactorized () const`

Say if matrix is factorized or not.

- virtual size_t [getColInd](#) (size_t i) const
Return Column index for column *i* (See the description for class [SpMatrix](#)).
- virtual size_t [getRowPtr](#) (size_t i) const
Return Row pointer for row *i* (See the description for class [SpMatrix](#)).
- T_ [operator\(\)](#) (size_t i) const
Operator () with one argument (Constant version).
- T_ & [operator\(\)](#) (size_t i)
Operator () with one argument (Non Constant version).
- T_ & [operator\[\]](#) (size_t k)
Operator [] (Non constant version).
- T_ [operator\[\]](#) (size_t k) const
Operator [] (Constant version).
- [Matrix](#) & [operator+=](#) (const [Matrix](#)< T_ > &m)
Operator +=.
- [Matrix](#) & [operator+=](#) (const T_ &x)
Operator +=.
- [Matrix](#) & [operator-=](#) (const [Matrix](#)< T_ > &m)
Operator -=.
- [Matrix](#) & [operator-=](#) (const T_ &x)
Operator -=.
- void [setPrintView](#) (size_t rmin, size_t rmax, size_t cmin, size_t cmax)
Set a window for matrix printing.
- void [getPrintView](#) (size_t &rmin, size_t &rmax, size_t &cmin, size_t &cmax) const
Return window data for matrix printing.

7.102.1 Detailed Description

template<class T_>class OFELI::TrMatrix< T_ >

To handle tridiagonal matrices.

This class enables storing and manipulating tridiagonal matrices. The template parameter is the type of matrix entries

Template Parameters

< T_ >	Data type (double, float, complex<double>, ...)
--------	---

7.102.2 Constructor & Destructor Documentation

TrMatrix ()

Default constructor.

Initialize a zero dimension tridiagonal matrix

7.102.3 Member Function Documentation

void setSize (size_t size)

Set size (number of rows) of matrix.

Parameters

<i>in</i>	<i>size</i>	Number of rows and columns.
-----------	-------------	-----------------------------

void Apxy (T_ a, const TrMatrix< T_ > & m)

Add to matrix the product of a matrix by a scalar.

Parameters

<i>in</i>	<i>a</i>	Scalar to premultiply
<i>in</i>	<i>m</i>	Matrix by which a is multiplied. The result is added to current instance

void Apxy (T_ a, const Matrix< T_ > * m) [virtual]

Add to matrix the product of a matrix by a scalar.

Parameters

<i>in</i>	<i>a</i>	Scalar to premultiply
<i>in</i>	<i>m</i>	Matrix by which a is multiplied. The result is added to current instance

Implements [Matrix< T_ >](#).**T_ operator() (size_t i, size_t j) const [virtual]**

Operator () (Constant version).

Parameters

<i>in</i>	<i>i</i>	Row index
<i>in</i>	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**T_ & operator() (size_t i, size_t j) [virtual]**

Operator () (Non constant version).

Parameters

<i>in</i>	<i>i</i>	Row index
<i>in</i>	<i>j</i>	Column index

Implements [Matrix< T_ >](#).**TrMatrix< T_ > & operator= (const TrMatrix< T_ > & m)**

Operator =.

Copy matrix *m* to current matrix instance.**TrMatrix< T_ > & operator= (const T_ & x)**

Operator =.

Assign constant value *x* to all matrix entries.**TrMatrix< T_ > & operator*= (const T_ & x)**

Operator *.

Premultiply matrix entries by constant value *x*.

int Solve (Vect< T_ > & b) [virtual]

Solve a linear system with current matrix (forward and back substitution).

Parameters

<i>in</i>	<i>b</i>	Vect instance that contains right-hand side on input and solution on output.
-----------	----------	--

Returns

- 0 if solution was normally performed,
- *n* if the *n*-th pivot is null.

Warning: Matrix is modified after this function.
Implements [Matrix< T_ >](#).

void setDiagonal (const class Mesh & mesh) [inherited]

Initialize matrix storage in the case where only diagonal terms are stored.

This member function is to be used for explicit time integration schemes

T_ getDiag (size_t k) const [inherited]

Return *k*-th diagonal entry of matrix.

First entry is given by **getDiag(1)**.

void Assembly (const class Element * el, T_ * a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a C-array.

Parameters

<i>in</i>	<i>el</i>	Pointer to element instance
<i>in</i>	<i>a</i>	Element matrix as a C-array

void Assembly (const class Element * el, const DMatrix< T_ > & a) [inherited]

Assembly of element matrix into global matrix.

Case where element matrix is given by a [DMatrix](#) instance.

Parameters

<i>in</i>	<i>el</i>	Pointer to element instance
<i>in</i>	<i>a</i>	Element matrix as a DMatrix instance

void Assembly (const class Side * sd, T_ * a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a C-array.

Parameters

<i>in</i>	<i>sd</i>	Pointer to side instance
<i>in</i>	<i>a</i>	Side matrix as a C-array instance

void Assembly (const class Side * sd, const DMatrix< T_ > & a) [inherited]

Assembly of side matrix into global matrix.

Case where side matrix is given by a [DMatrix](#) instance.

Parameters

in	<i>sd</i>	Pointer to side instance
in	<i>a</i>	Side matrix as a DMatrix instance

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (int dof, int code, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition to a given degree of freedom for a given code.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>dof</i>	Degree of freedom for which a boundary condition is to be enforced
in	<i>code</i>	Code for which a boundary condition is to be enforced
in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed valued at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (const class Mesh & mesh, Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in	<i>mesh</i>	Mesh instance from which information is extracted.
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (Vect< T_ > & b, int flag = 0) [inherited]

Impose by a penalty method a homegeneous (=0) essential boundary condition, using the [Mesh](#) instance provided by the constructor.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function **setPenal(..)**.

Parameters

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, const class Mesh & mesh, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0) [inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This gunction is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function **setPenal**.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in	<i>mesh</i>	Mesh instance from which information is extracted.

in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void Prescribe (size_t dof, Vect< T_ > & b, const Vect< T_ > & u, int flag = 0)
[inherited]

Impose by a penalty method an essential boundary condition when only one DOF is treated.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. This function is to be used if only one DOF per node is treated in the linear system. The penalty parameter is by default equal to 1.e20. It can be modified by member function `setPenal`.

Parameters

in	<i>dof</i>	Label of the concerned degree of freedom (DOF).
in,out	<i>b</i>	Vect instance that contains right-hand side.
in	<i>u</i>	Vect instance that contains imposed values at DOFs where they are to be imposed.
in	<i>flag</i>	Parameter to determine whether only the right-hand side is to be modified (dof>0) or both matrix and right-hand side (dof=0, default value).

void PrescribeSide (const class Mesh & mesh) [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function uses the [Mesh](#) instance provided by the constructor.

It modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

void PrescribeSide () [inherited]

Impose by a penalty method an essential boundary condition when DOFs are supported by sides.

This member function modifies diagonal terms in matrix and terms in vector that correspond to degrees of freedom with nonzero code in order to impose a boundary condition. The penalty parameter is defined by default equal to 1.e20. It can be modified by member function `setPenal(..)`.

int Solve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Solve system with factorized matrix (forward and back substitution).

Parameters

in	<i>b</i>	Vect instance that contains right-hand side
out	<i>x</i>	Vect instance that contains solution

Returns

- - 0 if solution was normally performed
 - n if the n -th pivot is null
- Solution is performed only if factorization has previously been invoked.

int FactorAndSolve (Vect< T_ > & b) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in,out	b	Vect instance that contains right-hand side on input and solution on output
--------	-----	---

int FactorAndSolve (const Vect< T_ > & b, Vect< T_ > & x) [inherited]

Factorize matrix and solve the linear system.

This is available only if the storage class enables it.

Parameters

in	b	Vect instance that contains right-hand side
out	x	Vect instance that contains solution

Returns

- 0 if solution was normally performed
- n if the n -th pivot is null

int isFactorized () const [inherited]

Say if matrix is factorized or not.

If the matrix was not factorized, the class does not allow solving by a direct solver.

T_ operator() (size_t i) const [inherited]

Operator () with one argument (Constant version).

Returns i -th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

in	i	entry index
----	-----	-------------

T_ & operator() (size_t i) [inherited]

Operator () with one argument (Non Constant version).

Returns i -th position in the array storing matrix entries. The first entry is at location 1. Entries are stored row by row.

Parameters

<i>in</i>	<i>i</i>	entry index
-----------	----------	-------------

T_ & operator[] (size_t k) [inherited]

Operator [] (Non constant version).

Returns k-th stored element in matrix Index k starts at 0.

T_ operator[] (size_t k) const [inherited]

Operator [] (Constant version).

Returns k-th stored element in matrix Index k starts at 0.

Matrix< T_ > & operator+= (const Matrix< T_ > & m) [inherited]

Operator +=.

Add matrix *m* to current matrix instance.**Matrix< T_ > & operator+= (const T_ & x)** [inherited]

Operator +=.

Add constant value *x* to all matrix entries.**Matrix< T_ > & operator-= (const Matrix< T_ > & m)** [inherited]

Operator -=.

Subtract matrix *m* from current matrix instance.**Matrix< T_ > & operator-= (const T_ & x)** [inherited]

Operator -=.

Subtract constant value *x* from all matrix entries.**void setPrintView (size_t rmin, size_t rmax, size_t cmin, size_t cmax)** [inherited]

Set a window for matrix printing.

Parameters

<i>in</i>	<i>rmin</i>	first row index to view
<i>in</i>	<i>rmax</i>	last row index to view
<i>in</i>	<i>cmin</i>	first column index to view
<i>in</i>	<i>cmax</i>	last column index to view

void getPrintView (size_t & rmin, size_t & rmax, size_t & cmin, size_t & cmax) const [inherited]

Return window data for matrix printing.

Parameters

<i>out</i>	<i>rmin</i>	first row index to view
------------	-------------	-------------------------

out	<i>rmax</i>	last row index to view
out	<i>cmin</i>	first column index to view
out	<i>cmax</i>	last column index to view

7.103 UserData< T_ > Class Template Reference

Abstract class to define by user various problem data.

Public Member Functions

- `UserData ()`
Default Constructor.
- `UserData (const class Mesh &mesh)`
Constructor using mesh instance.
- `virtual ~UserData ()`
Destructor.
- `void setTime (real_t time)`
Set time value.
- `void setDBC (Vect< T_ > &b)`
Set Dirichlet Boundary Conditions.
- `void setInitialData (Vect< T_ > &b)`
Set initial data.
- `void setBodyForce (Vect< T_ > &b)`
Set Nodewise Body Force using a Vect instance.
- `void setSurfaceForce (Vect< T_ > &b)`
Set Surface Force.
- `virtual T_ BoundaryCondition (const Point< real_t > &x, int code, real_t time=0., size_t dof=1)`
Define boundary condition to impose at point of coordinates x , with code $code$ at time $time$, for DOF dof
- `virtual T_ BodyForce (const Point< real_t > &x, real_t time=0., size_t dof=1)`
Define body force to impose at point of coordinates x , with code $code$ at time $time$, for DOF dof
- `virtual T_ SurfaceForce (const Point< real_t > &x, int code, real_t time=0., size_t dof=1)`
Define surface force to impose at point of coordinates x , with code $code$ at time $time$, for DOF dof
- `virtual T_ InitialData (const Point< real_t > &x, size_t dof=1)`
Define initial data to impose at point of coordinates x , for DOF dof

7.103.1 Detailed Description

`template<class T_>class OFELI::UserData< T_ >`

Abstract class to define by user various problem data.

The user has to implement a class that derives from the present one where the virtual functions are implemented.

Template Parameters

<code>< T_ ></code>	Data type (real_t, float, complex<real_t>, ...)
---------------------------	---

7.103.2 Member Function Documentation

void setBodyForce (Vect< T_ > & b)

Set Nodewise Body Force using a [Vect](#) instance.

Parameters

<code>in</code>	<code>b</code>	Vector containing body forces at nodes to impose
-----------------	----------------	--

virtual `T_ BoundaryCondition` (`const Point< real_t > & x`, `int code`, `real_t time = 0.`, `size_t dof = 1`) [virtual]

Define boundary condition to impose at point of coordinates `x`, with code `code` at time `time`, for DOF `dof`

Function to implement by user

virtual `T_ BodyForce` (`const Point< real_t > & x`, `real_t time = 0.`, `size_t dof = 1`) [virtual]

Define body force to impose at point of coordinates `x`, with code `code` at time `time`, for DOF `dof`

Function to implement by user

virtual `T_ SurfaceForce` (`const Point< real_t > & x`, `int code`, `real_t time = 0.`, `size_t dof = 1`) [virtual]

Define surface force to impose at point of coordinates `x`, with code `code` at time `time`, for DOF `dof`

Function to implement by user

virtual `T_ InitialData` (`const Point< real_t > & x`, `size_t dof = 1`) [virtual]

Define initial data to impose at point of coordinates `x`, for DOF `dof`

Function to implement by user

7.104 `Vect< T_ >` Class Template Reference

To handle general purpose vectors.

Inherits `vector< T_ >`.

Public Member Functions

- `Vect` ()
Default Constructor. Initialize a zero-length vector.
- `Vect` (`size_t n`)
Constructor setting vector size.
- `Vect` (`size_t nx`, `size_t ny`)
Constructor of a 2-D index vector.
- `Vect` (`size_t nx`, `size_t ny`, `size_t nz`)
Constructor of a 3-D index vector.
- `Vect` (`size_t n`, `T_ *x`)
Create an instance of class `Vect` as an image of a C/C++ array.
- `Vect` (`class Mesh &m`, `int nb_dof=0`, `int dof_type=NODE.FIELD`)
Constructor with a mesh instance.
- `Vect` (`class Mesh &m`, `string name`, `real_t t=0.0`, `int nb_dof=0`, `int dof_type=NODE.FIELD`)
Constructor with a mesh instance giving name and time for vector.
- `Vect` (`const class Element *el`, `const Vect< T_ > &v`)
Constructor of an element vector.

- `Vect` (const class `Side` *sd, const `Vect< T_ >` &v)
Constructor of a side vector.
- `Vect` (const `Vect< T_ >` &v, const `Vect< T_ >` &bc)
Constructor using boundary conditions.
- `Vect` (const `Vect< T_ >` &v, size_t nb_dof, size_t first_dof)
Constructor to select some components of a given vector.
- `Vect` (const `Vect< T_ >` &v)
Copy constructor.
- `Vect` (const `Vect< T_ >` &v, size_t n)
Constructor to select one component from a given 2 or 3-component vector.
- `Vect` (size_t d, const `Vect< T_ >` &v, const string &name=" ")
Constructor that extracts some degrees of freedom (components) from given instance of `Vect`.
- `~Vect` ()
Destructor.
- void `set` (const `T_` *v, size_t n)
Initialize vector with a c-array.
- void `select` (const `Vect< T_ >` &v, size_t nb_dof=0, size_t first_dof=1)
Initialize vector with another `Vect` instance.
- void `set` (const string &exp, size_t dof=1)
Initialize vector with an algebraic expression.
- void `set` (const class `Mesh` &ms, const string &exp, size_t dof=1)
Initialize vector with an algebraic expression with providing mesh data.
- void `set` (const `Vect< real_t >` &x, const string &exp)
Initialize vector with an algebraic expression.
- void `setMesh` (class `Mesh` &m, int nb_dof=0, int dof_type=NODE.FIELD)
Define mesh class to size vector.
- void `setSize` (size_t nx, size_t ny=1, size_t nz=1)
Set vector size (for 1-D, 2-D or 3-D cases)
- void `resize` (size_t n)
Set vector size.
- void `resize` (size_t n, `T_` v)
Set vector size and initialize to a constant value.
- void `setDOFType` (int dof_type)
Set DOF type of vector.
- size_t `size` () const
Return vector size.
- size_t `getNbDOF` () const
Return vector number of degrees of freedom.
- size_t `getNb` () const
Return vector number of entities (nodes, elements or sides)
- `Mesh` & `getMesh` () const
Return `Mesh` instance.
- bool `WithMesh` () const
Return true if vector contains a `Mesh` pointer, false if not.
- int `getDOFType` () const
- void `setTime` (`real_t` t)

- Set time value for vector.*

 - `real_t getTime ()` const
- Get time value for vector.*

 - void `setName` (string name)
- Set name of vector.*

 - string `getName ()` const
- Get name of vector.*

 - `real_t getNorm1 ()` const
- Calculate 1-norm of vector.*

 - `real_t getNorm2 ()` const
- Calculate 2-norm (Euclidean norm) of vector.*

 - `real_t getWNorm1 ()` const
- Calculate weighted 1-norm of vector The wighted 1-norm is the 1-Norm of the vector divided by its size.*

 - `real_t getWNorm2 ()` const
- Calculate weighted 2-norm of vector.*

 - `real_t getNormMax ()` const
- Calculate Max-norm (Infinite norm) of vector.*

 - `T_ getMin ()` const
- Calculate Min value of vector entries.*

 - `T_ getMax ()` const
- Calculate Max value of vector entries.*

 - size_t `getNx ()` const
- Return number of grid points in the x-direction if grid indexing is set.*

 - size_t `getNy ()` const
- Return number of grid points in the y-direction if grid indexing is set.*

 - size_t `getNz ()` const
- Return number of grid points in the z-direction if grid indexing is set.*

 - void `setNodeBC` (class `Mesh` &m, int code, `T_` val, size_t dof=1)
- Assign a given value to components of vector with given code.*

 - void `setNodeBC` (class `Mesh` &m, int code, const string &exp, size_t dof=1)
- Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.*

 - void `setNodeBC` (int code, `T_` val, size_t dof=1)
- Assign a given value to components of vector with given code.*

 - void `setNodeBC` (int code, const string &exp, size_t dof=1)
- Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.*

 - void `removeBC` (const class `Mesh` &ms, const `Vect< T_ >` &v, int dof=0)
- Remove boundary conditions.*

 - void `removeBC` (const `Vect< T_ >` &v, int dof=0)
- Remove boundary conditions.*

 - void `transferBC` (const `Vect< T_ >` &bc, int dof=0)
- Transfer boundary conditions to the vector.*

 - void `insertBC` (class `Mesh` &m, const `Vect< T_ >` &v, const `Vect< T_ >` &bc, int dof=0)
- Insert boundary conditions.*

 - void `insertBC` (class `Mesh` &m, const `Vect< T_ >` &v, int dof=0)

- Insert boundary conditions.*
- void `insertBC` (const `Vect`< T_ > &v, const `Vect`< T_ > &bc, int dof=0)
- Insert boundary conditions.*
- void `insertBC` (const `Vect`< T_ > &v, int dof=0)
- Insert boundary conditions.*
- void `Assembly` (const class `Element` *el, const `Vect`< T_ > &b)
- Assembly of element vector into current instance.*
- void `Assembly` (const class `Element` *el, const T_ *b)
- Assembly of element vector (as C-array) into Vect instance.*
- void `Assembly` (const class `Side` *sd, const `Vect`< T_ > &b)
- Assembly of side vector into Vect instance.*
- void `Assembly` (const class `Side` *sd, T_ *b)
- Assembly of side vector (as C-array) into Vect instance.*
- void `getGradient` (class `Vect`< T_ > &v)
- Evaluate the discrete Gradient vector of the current vector.*
- void `getGradient` (`Vect`< `Point`< T_ > > &v)
- Evaluate the discrete Gradient vector of the current vector.*
- void `getCurl` (`Vect`< T_ > &v)
- Evaluate the discrete curl vector of the current vector.*
- void `getCurl` (`Vect`< `Point`< T_ > > &v)
- Evaluate the discrete curl vector of the current vector.*
- void `getSCurl` (`Vect`< T_ > &v)
- Evaluate the discrete scalar curl in 2-D of the current vector.*
- void `getDivergence` (`Vect`< T_ > &v)
- Evaluate the discrete Divergence of the current vector.*
- `real_t` `getAverage` (const class `Element` &el, int type) const
- Return average value of vector in a given element.*
- void `save` (string file, int opt)
- Save vector in a file according to a given format.*
- `Vect`< T_ > & `MultAdd` (const `Vect`< T_ > &x, const T_ &a)
- Multiply by a constant then add to a vector.*
- void `Axpy` (T_ a, const `Vect`< T_ > &x)
- Add to vector the product of a vector by a scalar.*
- void `set` (size_t i, T_ val)
- Assign a value to an entry for a 1-D vector.*
- void `set` (size_t i, size_t j, T_ val)
- Assign a value to an entry for a 2-D vector.*
- void `set` (size_t i, size_t j, size_t k, T_ val)
- Assign a value to an entry for a 3-D vector.*
- void `add` (size_t i, T_ val)
- Add a value to an entry for a 1-index vector.*
- void `add` (size_t i, size_t j, T_ val)
- Add a value to an entry for a 2-index vector.*
- void `add` (size_t i, size_t j, size_t k, T_ val)
- Assign a value to an entry for a 3-index vector.*
- T_ & `operator()` (size_t i)

- *Operator () (Non constant version)*
- T_ [operator\(\)](#) (size_t i) const
- *Operator () (Constant version)*
- T_ & [operator\(\)](#) (size_t i, size_t j)
- *Operator () with 2-D indexing (Non constant version, case of a grid vector).*
- T_ [operator\(\)](#) (size_t i, size_t j) const
- *Operator () with 2-D indexing (Constant version).*
- T_ & [operator\(\)](#) (size_t i, size_t j, size_t k)
- *Operator () with 3-D indexing (Non constant version).*
- T_ [operator\(\)](#) (size_t i, size_t j, size_t k) const
- *Operator () with 3-D indexing (Constant version).*
- Vect< T_ > & [operator=](#) (const Vect< T_ > &v)
- *Operator = between vectors.*
- Vect< T_ > & [operator=](#) (const T_ &a)
- *Operator =*
- Vect< T_ > & [operator+=](#) (const Vect< T_ > &v)
- *Operator +=*
- Vect< T_ > & [operator+=](#) (const T_ &a)
- *Operator +=*
- Vect< T_ > & [operator-=](#) (const Vect< T_ > &v)
- *Operator -=*
- Vect< T_ > & [operator-=](#) (const T_ &a)
- *Operator -=*
- Vect< T_ > & [operator*=](#) (const T_ &a)
- *Operator *=*
- Vect< T_ > & [operator/=](#) (const T_ &a)
- *Operator /=*
- void [setPrintView](#) (size_t imin, size_t imax)
- *Set a window for vector printing.*
- void [getPrintView](#) (size_t &imin, size_t &imax) const
- *Return window data for vector printing.*
- const Mesh & [getMeshPtr](#) () const
- *Return reference to Mesh instance.*
- T_ [operator,](#) (const Vect< T_ > &v) const
- *Return Dot (scalar) product of two vectors.*

7.104.1 Detailed Description

template<class T_>class OFELI::Vect< T_ >

To handle general purpose vectors.

This template class enables defining and manipulating vectors of various data types. It inherits from the class `std::vector` An instance of class [Vect](#) can be:

- A simple vector of given size
- A vector with up to three indices, *i.e.*, an entry of the vector can be `a(i)`, `a(i,j)` or `a(i,j,k)`. This feature is useful, for instance, in the case of a structured grid

- A vector associate to a finite element mesh. In this case, a constructor uses a reference to the [Mesh](#) instance. The size of the vector is by default equal to the number of nodes x the number of degrees of freedom by node. If the degrees of freedom are supported by elements or sides, then the vector is sized accordingly

Operators =, [] and () are overloaded so that one can write for instance:

```
Vect<real_t> u(10), v(10);  
v = -1.0;  
u = v;  
u(3) = -2.0;
```

to set vector **v** entries to -1, copy vector **v** into vector **u** and assign third entry of **v** to -2. Note that entries of **v** are here **v(1)**, **v(2)**, ..., **v(10)**, i.e. vector entries start at index 1.

Template Parameters

< T_ >	Data type (real_t, float, complex<real_t>, ...)
--------	---

7.104.2 Constructor & Destructor Documentation

Vect (size_t n)

Constructor setting vector size.

Parameters

in	n	Size of vector
----	---	----------------

Vect (size_t nx, size_t ny)

Constructor of a 2-D index vector.

This constructor can be used for instance for a 2-D grid vector

Parameters

in	nx	Size for the first index
in	ny	Size for the second index

Remarks

The size of resulting vector is nx*ny

Vect (size_t nx, size_t ny, size_t nz)

Constructor of a 3-D index vector.

This constructor can be used for instance for a 3-D grid vector

Parameters

in	nx	Size for the first index
in	ny	Size for the second index
in	nz	Size for the third index

Remarks

The size of resulting vector is nx*ny*nz

Vect (size_t n, T_ * x)

Create an instance of class [Vect](#) as an image of a C/C++ array.

Parameters

in	<i>n</i>	Dimension of vector to construct
in	<i>x</i>	C-array to copy

Vect (class Mesh & m, int nb_dof = 0, int dof_type = NODE_FIELD)

Constructor with a mesh instance.

Parameters

in	<i>m</i>	Mesh instance
in	<i>nb_dof</i>	Number of degrees of freedom per node, element or side If nb_dof is set to 0 (default value) the constructor picks this number from the Mesh instance
in	<i>dof_type</i>	Type of degrees of freedom. To be given among the enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD (Default: NODE_FIELD)

Vect (class Mesh & m, string name, real_t t = 0.0, int nb_dof = 0, int dof_type = NODE_FIELD)

Constructor with a mesh instance giving name and time for vector.

Parameters

in	<i>m</i>	Mesh instance
in	<i>name</i>	Name of the vector
in	<i>t</i>	Time value for the vector
in	<i>nb_dof</i>	Number of degrees of freedom per node, element or side If nb_dof is set to 0 the constructor picks this number from the Mesh instance
in	<i>dof_type</i>	Type of degrees of freedom. To be given among the enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD (Default: NODE_FIELD)

Vect (const class Element * el, const Vect< T_ > & v)

Constructor of an element vector.

The constructed vector has local numbering of nodes

Parameters

in	<i>el</i>	Pointer to Element to localize
in	<i>v</i>	Global vector to localize

Vect (const class Side * sd, const Vect< T_ > & v)

Constructor of a side vector.

The constructed vector has local numbering of nodes

Parameters

in	<i>sd</i>	Pointer to Side to localize
----	-----------	---

in	v	Global vector to localize
----	---	---------------------------

Vect (const Vect< T_ > & v, const Vect< T_ > & bc)

Constructor using boundary conditions.

Boundary condition values contained in bc are reported to vector v

Parameters

in	v	Vect instance to update
in	bc	Vect instance containing imposed valued at desired DOF

Vect (const Vect< T_ > & v, size_t nb_dof, size_t first_dof)

Constructor to select some components of a given vector.

Parameters

in	v	Vect instance to extract from
in	nb_dof	Number of DOF to extract
in	first_dof	First DOF to extract For instance, a choice first_dof=2 and nb_dof=1 means that the second DOF of each node is copied in the vector

Vect (const Vect< T_ > & v, size_t n)

Constructor to select one component from a given 2 or 3-component vector.

Parameters

in	v	Vect instance to extract from
in	n	Component to extract (must be > 1 and < 4 or).

Vect (size_t d, const Vect< T_ > & v, const string & name = " ")

Constructor that extracts some degrees of freedom (components) from given instance of [Vect](#).

This constructor enables constructing a subvector of a given [Vect](#) instance. It selects a given list of degrees of freedom and put it according to a given order in the instance to construct.

Parameters

in	d	Integer number giving the list of degrees of freedom. This number is made of n digits where n is the number of degrees of freedom. Let us give an example: Assume that the instance v has 3 DOF by entity (node, element or side). The choice d=201 means that the constructed instance has 2 DOF where the first DOF is the third one of v, and the second DOF is the first one of f v. Consequently, no digit can be larger than the number of DOF the constructed instance. In this example, a choice d=103 would produce an error message.
in	v	Vect instance from which extraction is performed.
in	name	Name to assign to vector instance (Default value is " ").

Warning

Don't give zeros as first digits for the argument d. The number is in this case interpreted as octal !!

7.104.3 Member Function Documentation

void set (const T_ * v, size_t n)

Initialize vector with a c-array.

Parameters

in	<i>v</i>	c-array (pointer) to initialize Vect
in	<i>n</i>	size of array

void select (const Vect< T_ > & v, size_t nb_dof = 0, size_t first_dof = 1)

Initialize vector with another [Vect](#) instance.

Parameters

in	<i>v</i>	Vect instance to extract from
in	<i>nb_dof</i>	Number of DOF per node, element or side (By default, 0: Number of degrees of freedom extracted from the Mesh instance)
in	<i>first_dof</i>	First DOF to extract (Default: 1) For instance, a choice <i>first_dof</i> =2 and <i>nb_dof</i> =1 means that the second DOF of each node is copied in the vector

void set (const string & exp, size_t dof = 1)

Initialize vector with an algebraic expression.

Parameters

in	<i>exp</i>	Regular algebraic expression that defines a function of x, y and z which are coordinates of nodes.
in	<i>dof</i>	Degree of freedom to which the value is assigned [Default: 1]

void set (const class Mesh & ms, const string & exp, size_t dof = 1)

Initialize vector with an algebraic expression with providing mesh data.

Parameters

in	<i>ms</i>	Mesh instance
in	<i>exp</i>	Regular algebraic expression that defines a function of x, y and z which are coordinates of nodes.
in	<i>dof</i>	Degree of freedom to which the value is assigned [Default: 1]

void set (const Vect< real_t > & x, const string & exp)

Initialize vector with an algebraic expression.

Parameters

in	<i>x</i>	Vect instance that contains coordinates of points
in	<i>exp</i>	Regular algebraic expression that defines a function of x and i which are coordinates of nodes and indices starting from 1.

void setMesh (class Mesh & m, int nb_dof = 0, int dof_type = NODE_FIELD)

Define mesh class to size vector.

Parameters

in	<i>m</i>	Mesh instance
in	<i>nb_dof</i>	Number of degrees of freedom per node, element or side If <i>nb_dof</i> is set to 0 the constructor picks this number from the Mesh instance
in	<i>dof_type</i>	Parameter to precise the type of degrees of freedom. To be chosen among the enumerated values: <code>NODE_FIELD</code> , <code>ELEMENT_FIELD</code> , <code>SIDE_FIELD</code> , <code>EDGE_FIELD</code> [Default: <code>NODE_FIELD</code>]

void setSize (size_t nx, size_t ny = 1, size_t nz = 1)

Set vector size (for 1-D, 2-D or 3-D cases)

This function allocates memory for the vector but does not initialize its components

Parameters

in	<i>nx</i>	Number of grid points in x-direction
in	<i>ny</i>	Number of grid points in y-direction [Default: 1]
in	<i>nz</i>	Number of grid points in z-direction [Default: 1]

void resize (size_t n)

Set vector size.

This function allocates memory for the vector but does not initialize its components

Parameters

in	<i>n</i>	Size of vector
----	----------	----------------

void resize (size_t n, T_ v)

Set vector size and initialize to a constant value.

This function allocates memory for the vector

Parameters

in	<i>n</i>	Size of vector
in	<i>v</i>	Value to assign to vector entries

void setDOFType (int dof_type)

Set DOF type of vector.

The DOF type combined with number of DOF per component enable determining the size of vector

Parameters

in	<i>dof_type</i>	Type of degrees of freedom. Value to be chosen among the enumerated values: <code>NODE_FIELD</code> , <code>ELEMENT_FIELD</code> , <code>SIDE_FIELD</code> or <code>EDGE_FIELD</code>
----	-----------------	---

bool WithMesh () const

Return true if vector contains a [Mesh](#) pointer, false if not.

A [Vect](#) instance can be constructed using mesh information

int getDOFType () const

Return DOF type of vector

Returns

dof_type Type of degrees of freedom. Value among the enumerated values: NODE_FIELD, ELEMENT_FIELD, SIDE_FIELD or EDGE_FIELD

real_t getWNorm2 () const

Calculate weighted 2-norm of vector.

The weighted 2-norm is the 2-Norm of the vector divided by the square root of its size

void setNodeBC (class Mesh & m, int code, T_ val, size_t dof = 1)

Assign a given value to components of vector with given code.

Vector components are assumed nodewise

Parameters

in	<i>m</i>	Instance of mesh
in	<i>code</i>	Code for which nodes will be assigned prescribed value
in	<i>val</i>	Value to prescribe
in	<i>dof</i>	Degree of Freedom for which the value is assigned [default: 1]

void setNodeBC (class Mesh & m, int code, const string & exp, size_t dof = 1)

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise

Parameters

in	<i>m</i>	Instance of mesh
in	<i>code</i>	Code for which nodes will be assigned prescribed value
in	<i>exp</i>	Regular algebraic expression to prescribe
in	<i>dof</i>	Degree of Freedom for which the value is assigned [default: 1]

void setNodeBC (int code, T_ val, size_t dof = 1)

Assign a given value to components of vector with given code.

Vector components are assumed nodewise

Parameters

in	<i>code</i>	Code for which nodes will be assigned prescribed value
in	<i>val</i>	Value to prescribe
in	<i>dof</i>	Degree of Freedom for which the value is assigned [default: 1]

void setNodeBC (int code, const string & exp, size_t dof = 1)

Assign a given function (given by an interpretable algebraic expression) to components of vector with given code.

Vector components are assumed nodewise

Parameters

in	<i>code</i>	Code for which nodes will be assigned prescribed value
in	<i>exp</i>	Regular algebraic expression to prescribe
in	<i>dof</i>	Degree of Freedom for which the value is assigned [default: 1]

void removeBC (const class Mesh & ms, const Vect< T_ > & v, int dof = 0)

Remove boundary conditions.

This member function copies to current vector a vector where only non imposed DOF are retained.

Parameters

in	<i>ms</i>	Mesh instance
in	<i>v</i>	Vector (Vect instance to copy from)
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom

void removeBC (const Vect< T_ > & v, int dof = 0)

Remove boundary conditions.

This member function copies to current vector a vector where only non imposed DOF are retained.

Parameters

in	<i>v</i>	Vector (Vect instance to copy from)
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned [Default: 0] or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom.

Remarks

This function is to be used only when the [Vect](#) instance was constructed by using the [Mesh](#) instance

void transferBC (const Vect< T_ > & bc, int dof = 0)

Transfer boundary conditions to the vector.

Parameters

in	<i>bc</i>	Vect instance from which imposed degrees of freedom are copied to current instance
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom.

void insertBC (class Mesh & m, const Vect< T_ > & v, const Vect< T_ > & bc, int dof = 0)

Insert boundary conditions.

Parameters

in	<i>m</i>	Mesh instance.
in	<i>v</i>	Vect instance from which free degrees of freedom are copied to current instance.
in	<i>bc</i>	Vect instance from which imposed degrees of freedom are copied to current instance.
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom by node or side

void insertBC (class Mesh & m, const Vect< T_ > & v, int dof = 0)

Insert boundary conditions.

DOF with imposed boundary conditions are set to zero.

Parameters

in	<i>m</i>	Mesh instance.
in	<i>v</i>	Vect instance from which free degrees of freedom are copied to current instance.
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom by node or side

void insertBC (const Vect< T_ > & v, const Vect< T_ > & bc, int dof = 0)

Insert boundary conditions.

Parameters

in	<i>v</i>	Vect instance from which free degrees of freedom are copied to current instance.
in	<i>bc</i>	Vect instance from which imposed degrees of freedom are copied to current instance.
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom by node or side

void insertBC (const Vect< T_ > & v, int dof = 0)

Insert boundary conditions.

DOF with imposed boundary conditions are set to zero.

Parameters

in	<i>v</i>	Vect instance from which free degrees of freedom are copied to current instance.
in	<i>dof</i>	Parameter to say if all degrees of freedom are concerned (=0, Default) or if only one degree of freedom (<i>dof</i>) is inserted into vector <i>v</i> which has only one degree of freedom by node or side

void Assembly (const class Element * el, const Vect< T_ > & b)

Assembly of element vector into current instance.

Parameters

in	<i>el</i>	Pointer to element
in	<i>b</i>	Local vector to assemble (Instance of class Vect)

void Assembly (const class Element * el, const T_ * b)

Assembly of element vector (as C-array) into [Vect](#) instance.

Parameters

<code>in</code>	<code>el</code>	Pointer to element
<code>in</code>	<code>b</code>	Local vector to assemble (C-Array)

void Assembly (const class Side * sd, const Vect< T_ > & b)

Assembly of side vector into [Vect](#) instance.

Parameters

<code>in</code>	<code>sd</code>	Pointer to side
<code>in</code>	<code>b</code>	Local vector to assemble (Instance of class Vect)

void Assembly (const class Side * sd, T_ * b)

Assembly of side vector (as C-array) into [Vect](#) instance.

Parameters

<code>in</code>	<code>sd</code>	Pointer to side
<code>in</code>	<code>b</code>	Local vector to assemble (C-Array)

void getGradient (class Vect< T_ > & v)

Evaluate the discrete Gradient vector of the current vector.

The resulting gradient is stored in a [Vect](#) instance This function handles node vectors assuming P_1 approximation The gradient is then a constant vector for each element.

Parameters

<code>in</code>	<code>v</code>	Vect instance that contains the gradient, where $v(n,1)$, $v(n,2)$ and $v(n,3)$ are respectively the x and y and z derivatives at element n .
-----------------	----------------	--

void getGradient (Vect< Point< T_ > > & v)

Evaluate the discrete Gradient vector of the current vector.

The resulting gradient is stored in an [Vect](#) instance This function handles node vectors assuming P_1 approximation The gradient is then a constant vector for each element.

Parameters

<code>in</code>	<code>v</code>	Vect instance that contains the gradient, where $v(n,1).x$, $v(n,2).y$ and $v(n,3).z$ are respectively the x and y and z derivatives at element n .
-----------------	----------------	--

void getCurl (Vect< T_ > & v)

Evaluate the discrete curl vector of the current vector.

The resulting curl is stored in a [Vect](#) instance This function handles node vectors assuming P_1 approximation The curl is then a constant vector for each element.

Parameters

<code>in</code>	<code>v</code>	Vect instance that contains the curl, where $v(n,1)$, $v(n,2)$ and $v(n,3)$ are respectively the x and y and z curl components at element n .
-----------------	----------------	--

void getCurl (Vect< Point< T_ > > & v)

Evaluate the discrete curl vector of the current vector.

The resulting curl is stored in a [Vect](#) instance This function handles node vectors assuming P_1 approximation The curl is then a constant vector for each element.

Parameters

<i>in</i>	<i>v</i>	Vect instance that contains the curl, where $v(n,1).x$, $v(n,2).y$ and $v(n,3).z$ are respectively the x and y and z curl components at element n .
-----------	----------	--

void getSCurl (Vect< T_ > & v)

Evaluate the discrete scalar curl in 2-D of the current vector.

The resulting curl is stored in a [Vect](#) instance This function handles node vectors assuming P_1 approximation The curl is then a constant vector for each element.

Parameters

<i>in</i>	<i>v</i>	Vect instance that contains the scalar curl.
-----------	----------	--

void getDivergence (Vect< T_ > & v)

Evaluate the discrete Divergence of the current vector.

The resulting divergence is stored in a [Vect](#) instance This function handles node vectors assuming P_1 approximation The divergence is then a constant vector for each element.

Parameters

<i>in</i>	<i>v</i>	Vect instance that contains the divergence.
-----------	----------	---

real.t getAverage (const class Element & el, int type) const

Return average value of vector in a given element.

Parameters

<i>in</i>	<i>el</i>	Element instance
<i>in</i>	<i>type</i>	Type of element. This is to be chosen among enumerated values: LINE2, TRIANG3, QUAD4, TETRA4, HEXA8, PENTA6

void save (string file, int opt)

Save vector in a file according to a given format.

Parameters

<i>in</i>	<i>file</i>	Output file where to save the vector
<i>in</i>	<i>opt</i>	Option to choose file format to save. This is to be chosen among enumerated values: GMSH, GNUPLOT, MATLAB, TECPLOT and VTK

Vect<T_>& MultAdd (const Vect< T_ > & x, const T_ & a)

Multiply by a constant then add to a vector.

Parameters

<code>in</code>	<code>x</code>	Vect instance to add
<code>in</code>	<code>a</code>	Constant to multiply before adding

void Apxy (T_ a, const Vect< T_ > & x)

Add to vector the product of a vector by a scalar.

Parameters

<code>in</code>	<code>a</code>	Scalar to premultiply
<code>in</code>	<code>x</code>	Vect instance by which a is multiplied. The result is added to current instance

void set (size_t i, T_ val)

Assign a value to an entry for a 1-D vector.

Parameters

<code>in</code>	<code>i</code>	Rank index in vector (starts at 1)
<code>in</code>	<code>val</code>	Value to assign

void set (size_t i, size_t j, T_ val)

Assign a value to an entry for a 2-D vector.

Parameters

<code>in</code>	<code>i</code>	First index in vector (starts at 1)
<code>in</code>	<code>j</code>	Second index in vector (starts at 1)
<code>in</code>	<code>val</code>	Value to assign

void set (size_t i, size_t j, size_t k, T_ val)

Assign a value to an entry for a 3-D vector.

Parameters

<code>in</code>	<code>i</code>	First index in vector (starts at 1)
<code>in</code>	<code>j</code>	Second index in vector (starts at 1)
<code>in</code>	<code>k</code>	Third index in vector (starts at 1)
<code>in</code>	<code>val</code>	Value to assign

void add (size_t i, T_ val)

Add a value to an entry for a 1-index vector.

Parameters

<code>in</code>	<code>i</code>	Rank index in vector (starts at 1)
<code>in</code>	<code>val</code>	Value to assign

void add (size_t i, size_t j, T_ val)

Add a value to an entry for a 2-index vector.

Parameters

in	<i>i</i>	First index in vector (starts at 1)
in	<i>j</i>	Second index in vector (starts at 1)
in	<i>val</i>	Value to assign

void add (size_t i, size_t j, size_t k, T_ val)

Assign a value to an entry for a 3-index vector.

Parameters

in	<i>i</i>	First index in vector (starts at 1)
in	<i>j</i>	Second index in vector (starts at 1)
in	<i>k</i>	Third index in vector (starts at 1)
in	<i>val</i>	Value to assign

T_& operator() (size_t i)

Operator () (Non constant version)

Parameters

in	<i>i</i>	Rank index in vector (starts at 1) <ul style="list-style-type: none"> • v(i) starts at v(1) to v(size()) • v(i) is the same element as v[i-1]
----	----------	---

T_ operator() (size_t i) const

Operator () (Constant version)

Parameters

in	<i>i</i>	Rank index in vector (starts at 1) <ul style="list-style-type: none"> • v(i) starts at v(1) to v(size()) • v(i) is the same element as v[i-1]
----	----------	---

T_& operator() (size_t i, size_t j)

Operator () with 2-D indexing (Non constant version, case of a grid vector).

Parameters

in	<i>i</i>	first index in vector (Number of vector components in the x-grid)
in	<i>j</i>	second index in vector (Number of vector components in the y-grid) v(i,j) starts at v(1,1) to v(getNx() , getNy())

T_ operator() (size_t i, size_t j) const

Operator () with 2-D indexing (Constant version).

Parameters

in	<i>i</i>	first index in vector (Number of vector components in the x-grid)
in	<i>j</i>	second index in vector (Number of vector components in the y-grid) v(<i>i</i> , <i>j</i>) starts at v(1,1) to v(getNx() , getNy())

T_ operator() (size_t i, size_t j, size_t k)

Operator () with 3-D indexing (Non constant version).

Parameters

in	<i>i</i>	first index in vector (Number of vector components in the x-grid)
in	<i>j</i>	second index in vector (Number of vector components in the y-grid)
in	<i>k</i>	third index in vector (Number of vector components in the z-grid) v(<i>i</i> , <i>j</i> , <i>k</i>) starts at v(1,1,1) to v(getNx() , getNy() , getNz())

T_ operator() (size_t i, size_t j, size_t k) const

Operator () with 3-D indexing (Constant version).

Parameters

in	<i>i</i>	first index in vector (Number of vector components in the x-grid)
in	<i>j</i>	second index in vector (Number of vector components in the y-grid)
in	<i>k</i>	third index in vector (Number of vector components in the z-grid) v(<i>i</i> , <i>j</i> , <i>k</i>) starts at v(1,1,1) to v(getNx() , getNy() , getNz())

Vect<T_>& operator= (const T_ & a)

Operator =

Assign a constant to vector entries

Parameters

in	<i>a</i>	Value to set
----	----------	--------------

Vect<T_>& operator+= (const Vect< T_ > & v)

Operator +=

Add vector x to current vector instance.

Parameters

in	<i>v</i>	Vect instance to add to instance
----	----------	--

Vect<T_>& operator+= (const T_ & a)

Operator +=

Add a constant to current vector entries.

Parameters

in	<i>a</i>	Value to add to vector entries
----	----------	--------------------------------

Vect<T_>& operator-= (const Vect< T_ > & v)

Operator -=

Parameters

in	v	Vect instance to subtract from
----	---	--

Vect<T_>& operator-= (const T_ & a)

Operator -=

Subtract constant from vector entries.

Parameters

in	a	Value to subtract from
----	---	------------------------

Vect<T_>& operator*= (const T_ & a)

Operator *=

Parameters

in	a	Value to multiply by
----	---	----------------------

Vect<T_>& operator/= (const T_ & a)

Operator /=

Parameters

in	a	Value to divide by
----	---	--------------------

void setPrintView (size_t imin, size_t imax)

Set a window for vector printing.

Parameters

in	imin	first vector index to view
in	imax	last vector index to view

void getPrintView (size_t & imin, size_t & imax) const

Return window data for vector printing.

Parameters

out	imin	first vector index to view
out	imax	last vector index to view

T_ operator, (const Vect< T_ > & v) const

Return Dot (scalar) product of two vectors.

A typical use of this operator is `double a = (v,w)` where v and w are 2 instances of [Vect<double>](#)

Parameters

in	v	Vect instance by which the current instance is multiplied
----	---	---

Index

- A
 - OFELI, [111](#)
- AB2
 - OFELI, [109](#)
- ADAMS_BASHFORTH
 - OFELI, [109](#)
- AUX_INPUT_FIELD_1
 - OFELI, [108](#)
- AUX_INPUT_FIELD_2
 - OFELI, [109](#)
- AUX_INPUT_FIELD_3
 - OFELI, [109](#)
- AUX_INPUT_FIELD_4
 - OFELI, [109](#)
- AbsEqua< T_ >, [113](#)
- Add
 - OFELI::Element, [388](#), [389](#)
 - OFELI::Mesh, [667](#), [670](#)
 - OFELI::Side, [743](#)
- add
 - OFELI::DMatrix, [270](#)
 - OFELI::DSMatrix, [286](#)
 - OFELI::SkMatrix, [752](#)
 - OFELI::SkSMatrix, [767](#)
 - OFELI::SpMatrix, [787](#)
 - OFELI::Vect, [853](#), [854](#)
- AddMidNodes
 - OFELI::Mesh, [674](#)
- AnalysisType
 - OFELI, [110](#)
- ArrayType
 - OFELI, [109](#)
- Assembly
 - OFELI::BMatrix, [157](#), [158](#)
 - OFELI::DMatrix, [273](#), [274](#)
 - OFELI::DSMatrix, [288](#), [290](#)
 - OFELI::Matrix, [651](#), [652](#)
 - OFELI::SkMatrix, [755](#), [756](#)
 - OFELI::SkSMatrix, [769](#), [770](#)
 - OFELI::SpMatrix, [789](#), [790](#)
 - OFELI::TrMatrix, [829](#), [830](#)
 - OFELI::Vect, [850](#), [851](#)
- Axb
 - OFELI::Laplace2DT3, [602](#)
- AxbAssembly
 - OFELI::Bar2DL2, [130](#)
 - OFELI::Beam3DL2, [145](#), [146](#)
 - OFELI::DC1DL2, [184](#), [185](#)
 - OFELI::DC2DT3, [205](#), [206](#)
 - OFELI::DC2DT6, [223](#)
 - OFELI::DC3DAT3, [242](#), [243](#)
 - OFELI::DC3DT4, [262](#), [263](#)
 - OFELI::EC2D1T3, [306](#), [307](#)
 - OFELI::Elas2DQ4, [330](#), [331](#)
 - OFELI::Elas2DT3, [350](#), [351](#)
 - OFELI::Elas3DH8, [367](#)
 - OFELI::Elas3DT4, [382](#)
 - OFELI::Equa_Electromagnetics, [404](#)
 - OFELI::Equa_Fluid, [416](#), [417](#)
 - OFELI::Equa_Laplace, [429](#), [430](#)
 - OFELI::Equa_Solid, [445](#)
 - OFELI::Equa_Therm, [459](#)
 - OFELI::Equation, [472](#)
 - OFELI::HelmholtzBT3, [505](#), [506](#)
 - OFELI::Laplace1DL2, [555](#), [556](#)
 - OFELI::Laplace1DL3, [568](#), [569](#)
 - OFELI::Laplace2DFVT, [581](#), [582](#)
 - OFELI::Laplace2DMHRT0, [595](#)
 - OFELI::Laplace2DT3, [609](#), [610](#)
 - OFELI::NSP2DQ41, [714](#)
 - OFELI::TINS2DT3B, [813](#)
- Axpy
 - OFELI::BMatrix, [156](#)
 - OFELI::DMatrix, [271](#)
 - OFELI::DSMatrix, [287](#)
 - OFELI::Matrix, [651](#)
 - OFELI::SkMatrix, [751](#)
 - OFELI::SkSMatrix, [766](#)
 - OFELI::SpMatrix, [786](#)
 - OFELI::TrMatrix, [828](#)
 - OFELI::Vect, [853](#)
 - Utilities, [82](#)
- b
 - OFELI, [111](#)
- BACKWARD_EULER
 - OFELI, [109](#)
- BICG_SOLVER
 - OFELI, [110](#)
- BICG_STAB_SOLVER

OFELI, [110](#)
 BODY_FORCE
 OFELI, [108](#)
 BOUNDARY_CONDITION
 OFELI, [108](#)
 BOUNDARY_FORCE
 OFELI, [108](#)
 BOUNDARY_TRACTION
 OFELI, [108](#)
 BUOYANCY
 OFELI, [108](#)
 BCAsConstraint
 Solver, [54](#)
 BCType
 OFELI, [111](#)
 BMatrix
 OFELI::BMatrix, [155](#)
 BMatrix< T_ >, [152](#)
 BSpline
 Utilities, [80](#)
 banner
 Utilities, [81](#)
 Bar2DL2, [116](#)
 OFELI::Bar2DL2, [121](#)
 Beam3DL2, [132](#)
 OFELI::Beam3DL2, [137](#), [138](#)
 BiCG
 Solver, [49](#)
 BiCGStab
 Solver, [50](#)
 BiotSavart, [148](#)
 OFELI::BiotSavart, [149](#)
 BodyForce
 OFELI::UserData, [836](#)
 BodyRHS
 OFELI::Bar2DL2, [122](#)
 OFELI::DC1DL2, [175](#), [176](#)
 OFELI::DC2DT3, [196](#)
 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [233](#), [234](#)
 OFELI::DC3DT4, [253](#)
 OFELI::Elas2DQ4, [322](#)
 OFELI::Elas2DT3, [341](#)
 OFELI::Elas3DH8, [357](#), [359](#)
 OFELI::Elas3DT4, [373](#)
 OFELI::Laplace1DL2, [548](#)
 OFELI::Laplace1DL3, [561](#)
 OFELI::Laplace2DT3, [601](#)
 OFELI::NSP2DQ41, [707](#)
 BoundaryCondition
 OFELI::UserData, [836](#)
 BoundaryRHS
 OFELI::DC1DL2, [176](#)
 OFELI::DC2DT3, [196](#), [197](#)

 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [234](#)
 OFELI::DC3DT4, [254](#)
 OFELI::Elas2DQ4, [322](#)
 OFELI::Elas2DT3, [341](#)
 OFELI::Elas3DH8, [357](#)
 OFELI::Elas3DT4, [374](#)
 OFELI::Laplace1DL2, [548](#)
 OFELI::Laplace1DL3, [561](#)
 OFELI::Laplace2DT3, [601](#)
 OFELI::NSP2DQ41, [707](#)
 Bput
 OFELI::Mesh, [675](#)
 Brick, [163](#)
 OFELI::Brick, [164](#)
 build
 OFELI::DC2DT6, [216](#)
 OFELI::Equa_Therm, [452](#)
 OFELI::Laplace2DMHRT0, [588](#)
 OFELI::Laplace2DT3, [602](#)
 buildEigen
 OFELI::Bar2DL2, [123](#)
 OFELI::Beam3DL2, [138](#)
 OFELI::DC1DL2, [177](#)
 OFELI::DC2DT3, [198](#), [199](#)
 OFELI::DC2DT6, [216](#)
 OFELI::DC3DAT3, [234](#), [236](#)
 OFELI::DC3DT4, [255](#)
 OFELI::Elas2DQ4, [323](#)
 OFELI::Elas2DT3, [343](#), [344](#)
 OFELI::Elas3DH8, [359](#)
 OFELI::Elas3DT4, [374](#)
 OFELI::Equa_Solid, [437](#)
 OFELI::Equa_Therm, [452](#)
 OFELI::Laplace2DT3, [602](#)
 CAPACITY
 OFELI, [108](#)
 CG_SOLVER
 OFELI, [110](#)
 CGS_SOLVER
 OFELI, [110](#)
 CONTACT
 OFELI, [108](#)
 CONTACT_BC
 OFELI, [111](#)
 CONVECTION
 OFELI, [108](#)
 CRANK_NICOLSON
 OFELI, [109](#)
 CG
 Solver, [51](#)
 CGS
 Solver, [51](#)

Capacity
 OFELI::DC1DL2, [174](#)
 OFELI::DC2DT3, [194](#)
 OFELI::DC3DAT3, [233](#)
 OFELI::DC3DT4, [252](#)

CapacityToLHS
 OFELI::DC1DL2, [174](#)
 OFELI::DC2DT3, [194](#)
 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [232](#)
 OFELI::DC3DT4, [252](#)
 OFELI::Equa_Therm, [451](#)

CapacityToRHS
 OFELI::DC1DL2, [174](#)
 OFELI::DC2DT3, [194](#)
 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [233](#)
 OFELI::DC3DT4, [252](#)
 OFELI::Equa_Therm, [451](#)

check
 OFELI::Iter, [543](#)
 OFELI::Line2H, [633](#)
 OFELI::Material, [648](#)
 OFELI::Triang3, [817](#)

check_error
 OFELI::FMM2D, [481](#)
 OFELI::FMM3D, [483](#)
 OFELI::FMMSolver, [484](#)

checkDelaunay
 OFELI::Laplace2DFVT, [574](#)

checkSturm
 OFELI::Eigen, [315](#)

Circle, [165](#)
 OFELI::Circle, [166](#)

Clear
 Utilities, [82](#)

Conservation Law Equations, [15](#)

ContactPressure
 OFELI::Elas2DT3, [343](#)

Contains
 OFELI::Element, [389](#)
 OFELI::Side, [745](#)

Convection
 OFELI::DC1DL2, [175](#)
 OFELI::DC2DT3, [195](#)
 OFELI::DC2DT6, [214](#)
 OFELI::DC3DT4, [253](#)

ConvectionToRHS
 OFELI::DC1DL2, [175](#)
 OFELI::DC2DT3, [195](#)

Converged
 Global Variables, [37](#)

createBoundarySideList
 OFELI::Mesh, [669](#)

createInternalSideList
 OFELI::Mesh, [669](#)

CurlEdgeSh
 OFELI::Tetra4, [800](#)

DEVIATORIC
 OFELI, [108](#)

DIAG_PREC
 OFELI, [111](#)

DIAGONAL
 OFELI, [110](#)

DIFFUSION
 OFELI, [108](#)

DIFFUSION_CONVECTION
 OFELI, [109](#)

DILATATION
 OFELI, [108](#)

DIRECT_SOLVER
 OFELI, [110](#)

DISPLACEMENT_FIELD
 OFELI, [109](#)

DC1DL2, [168](#)
 OFELI::DC1DL2, [173](#)

DC2DT3, [187](#)
 OFELI::DC2DT3, [192](#), [193](#)

DC2DT6, [208](#)
 OFELI::DC2DT6, [212](#), [213](#)

DC3DAT3, [225](#)
 OFELI::DC3DAT3, [230](#), [232](#)

DC3DT4, [245](#)
 OFELI::DC3DT4, [250](#), [251](#)

DGELEMENTAssembly
 OFELI::Bar2DL2, [128](#)
 OFELI::Beam3DL2, [143](#), [144](#)
 OFELI::DC1DL2, [182](#), [183](#)
 OFELI::DC2DT3, [203](#), [204](#)
 OFELI::DC2DT6, [221](#), [222](#)
 OFELI::DC3DAT3, [240](#), [241](#)
 OFELI::DC3DT4, [260](#), [261](#)
 OFELI::EC2D1T3, [304](#), [305](#)
 OFELI::Elas2DQ4, [328](#), [329](#)
 OFELI::Elas2DT3, [348](#), [349](#)
 OFELI::Elas3DH8, [365](#)
 OFELI::Elas3DT4, [380](#)
 OFELI::Equa_Electromagnetics, [402](#)
 OFELI::Equa_Fluid, [414](#), [415](#)
 OFELI::Equa_Laplace, [427](#), [428](#)
 OFELI::Equa_Solid, [443](#)
 OFELI::Equa_Therm, [457](#), [458](#)
 OFELI::Equation, [469](#), [470](#)
 OFELI::HelmholtzBT3, [503](#), [504](#)
 OFELI::Laplace1DL2, [553](#), [554](#)
 OFELI::Laplace1DL3, [566](#), [567](#)
 OFELI::Laplace2DFVT, [579](#), [580](#)

OFELI::Laplace2DMHRT0, 593
 OFELI::Laplace2DT3, 607, 608
 OFELI::NSP2DQ41, 712, 713
 OFELI::TINS2DT3B, 811, 812
 dLine
 OFELI::Brick, 165
 OFELI::Circle, 168
 OFELI::Ellipse, 394
 OFELI::Figure, 480
 OFELI::Polygon, 728
 OFELI::Rectangle, 739
 OFELI::Sphere, 777
 DMatrix
 OFELI::DMatrix, 268, 269
 DMatrix< T_ >, 265
 DOF
 OFELI::Edge, 310
 OFELI::Node, 699
 OFELI::Side, 742
 DP1toP1
 OFELI::Reconstruction, 737
 DSMatrix
 OFELI::DSMatrix, 285, 286
 DSMatrix< T_ >, 282
 DSh
 OFELI::FEShape, 478
 OFELI::Hexa8, 509
 OFELI::Line2, 631
 OFELI::Line2H, 633
 OFELI::Penta6, 719
 OFELI::Quad4, 735
 OFELI::Tetra4, 799
 OFELI::Triang6S, 819, 820
 OFELI::triangle, 822, 824
 Deactivate
 OFELI::Grid, 493
 DeformMesh
 Finite Element Mesh, 23
 Delete
 OFELI::Mesh, 671, 672
 DeleteElement
 OFELI::Mesh, 671
 DeleteNode
 OFELI::Mesh, 671
 DeleteSide
 OFELI::Mesh, 671
 Deviator
 OFELI::Elas2DT3, 340
 DeviatorToRHS
 OFELI::Elas2DT3, 340
 DiagBC
 OFELI::Bar2DL2, 124
 OFELI::Beam3DL2, 139
 OFELI::DC1DL2, 178

OFELI::DC2DT3, 199
 OFELI::DC2DT6, 217
 OFELI::DC3DAT3, 236
 OFELI::DC3DT4, 255
 OFELI::EC2D1T3, 300
 OFELI::Elas2DQ4, 324
 OFELI::Elas2DT3, 344
 OFELI::Elas3DH8, 360
 OFELI::Elas3DT4, 375
 OFELI::Equa_Electromagnetics, 398
 OFELI::Equa_Fluid, 409
 OFELI::Equa_Laplace, 422
 OFELI::Equa_Solid, 438
 OFELI::Equa_Therm, 453
 OFELI::Equation, 466
 OFELI::HelmholtzBT3, 499
 OFELI::Laplace1DL2, 549
 OFELI::Laplace1DL3, 562
 OFELI::Laplace2DFVT, 575
 OFELI::Laplace2DMHRT0, 589
 OFELI::Laplace2DT3, 603
 OFELI::NSP2DQ41, 708
 OFELI::TINS2DT3B, 807
 DiagPrescribe
 OFELI::SkMatrix, 753
 OFELI::SpMatrix, 784
 Diffusion
 OFELI::DC1DL2, 174
 OFELI::DC2DT3, 194
 OFELI::DC2DT6, 214
 OFELI::DC3DAT3, 233
 OFELI::DC3DT4, 252
 DiffusionToRHS
 OFELI::DC1DL2, 174
 OFELI::DC2DT3, 195
 OFELI::DC3DAT3, 233
 OFELI::DC3DT4, 252
 Dilatation
 OFELI::Elas2DT3, 340
 DilatationToRHS
 OFELI::Elas2DQ4, 322
 OFELI::Elas2DT3, 341
 Domain, 278
 OFELI::Domain, 280
 Dot
 Utilities, 82
 Vector and Matrix, 87
 E2T
 OFELI::PhaseChange, 720
 ELECTRIC
 OFELI, 108
 EXTERNAL_BOUNDARY
 OFELI::Side, 742

EC2D1T3, [295](#)
 OFELI::EC2D1T3, [298](#), [299](#)
 EVAL
 Utilities, [70](#)
 Edge, [309](#)
 OFELI::Edge, [310](#)
 EdgeList, [311](#)
 EdgeSh
 OFELI::Tetra4, [799](#)
 EdgesAreDOF
 OFELI::Mesh, [675](#)
 Eigen, [312](#)
 OFELI::Eigen, [313](#), [314](#)
 Elas2DQ4, [316](#)
 OFELI::Elas2DQ4, [321](#)
 Elas2DT3, [333](#)
 OFELI::Elas2DT3, [338](#), [339](#)
 Elas3DH8, [353](#)
 OFELI::Elas3DH8, [357](#)
 Elas3DT4, [369](#)
 Electric
 OFELI::EC2D1T3, [299](#)
 Electromagnetics, [16](#)
 Element, [384](#)
 OFELI::Element, [386](#), [387](#)
 element_assembly
 General Purpose Equations, [32](#), [33](#)
 ElementAssembly
 OFELI::Bar2DL2, [126](#), [127](#)
 OFELI::Beam3DL2, [142](#), [143](#)
 OFELI::DC1DL2, [181](#), [182](#)
 OFELI::DC2DT3, [202](#), [203](#)
 OFELI::DC2DT6, [220](#), [221](#)
 OFELI::DC3DAT3, [239](#), [240](#)
 OFELI::DC3DT4, [259](#), [260](#)
 OFELI::EC2D1T3, [303](#), [304](#)
 OFELI::Elas2DQ4, [327](#), [328](#)
 OFELI::Elas2DT3, [347](#), [348](#)
 OFELI::Elas3DH8, [362](#), [364](#)
 OFELI::Elas3DT4, [377](#), [379](#)
 OFELI::Equa_Electromagnetics, [400](#), [401](#)
 OFELI::Equa_Fluid, [413](#), [414](#)
 OFELI::Equa_Laplace, [426](#), [427](#)
 OFELI::Equa_Solid, [440](#), [442](#)
 OFELI::Equa_Therm, [456](#), [457](#)
 OFELI::Equation, [468](#), [469](#), [471](#)
 OFELI::HelmholtzBT3, [502](#), [503](#)
 OFELI::Laplace1DL2, [552](#), [553](#)
 OFELI::Laplace1DL3, [565](#), [566](#)
 OFELI::Laplace2DFVT, [578](#), [579](#)
 OFELI::Laplace2DMHRT0, [591](#), [592](#)
 OFELI::Laplace2DT3, [606](#), [607](#)
 OFELI::NSP2DQ41, [711](#), [712](#)
 OFELI::TINS2DT3B, [810](#), [811](#)

ElementList, [391](#)
 ElementNodeCoordinates
 OFELI::Bar2DL2, [126](#)
 OFELI::Beam3DL2, [142](#)
 OFELI::DC1DL2, [181](#)
 OFELI::DC2DT3, [202](#)
 OFELI::DC2DT6, [219](#)
 OFELI::DC3DAT3, [239](#)
 OFELI::DC3DT4, [259](#)
 OFELI::EC2D1T3, [303](#)
 OFELI::Elas2DQ4, [327](#)
 OFELI::Elas2DT3, [347](#)
 OFELI::Elas3DH8, [362](#)
 OFELI::Elas3DT4, [377](#)
 OFELI::Equa_Electromagnetics, [400](#)
 OFELI::Equa_Fluid, [413](#)
 OFELI::Equa_Laplace, [426](#)
 OFELI::Equa_Solid, [440](#)
 OFELI::Equa_Therm, [455](#)
 OFELI::Equation, [468](#)
 OFELI::HelmholtzBT3, [502](#)
 OFELI::Laplace1DL2, [552](#)
 OFELI::Laplace1DL3, [565](#)
 OFELI::Laplace2DFVT, [578](#)
 OFELI::Laplace2DMHRT0, [591](#)
 OFELI::Laplace2DT3, [606](#)
 OFELI::NSP2DQ41, [710](#)
 OFELI::TINS2DT3B, [809](#)
 ElementNodeVector
 OFELI::Bar2DL2, [124](#)
 OFELI::Beam3DL2, [140](#)
 OFELI::DC1DL2, [179](#)
 OFELI::DC2DT3, [200](#)
 OFELI::DC2DT6, [217](#), [218](#)
 OFELI::DC3DAT3, [237](#)
 OFELI::DC3DT4, [256](#)
 OFELI::EC2D1T3, [301](#)
 OFELI::Elas2DQ4, [325](#)
 OFELI::Elas2DT3, [345](#)
 OFELI::Elas3DH8, [360](#), [361](#)
 OFELI::Elas3DT4, [375](#), [376](#)
 OFELI::Equa_Electromagnetics, [398](#)
 OFELI::Equa_Fluid, [410](#)
 OFELI::Equa_Laplace, [423](#)
 OFELI::Equa_Solid, [438](#), [439](#)
 OFELI::Equa_Therm, [453](#), [454](#)
 OFELI::Equation, [466](#), [467](#)
 OFELI::HelmholtzBT3, [500](#)
 OFELI::Laplace1DL2, [550](#)
 OFELI::Laplace1DL3, [563](#)
 OFELI::Laplace2DFVT, [576](#)
 OFELI::Laplace2DMHRT0, [589](#)
 OFELI::Laplace2DT3, [604](#)
 OFELI::NSP2DQ41, [708](#), [709](#)

OFELI::TINS2DT3B, [807](#), [808](#)
 ElementNodeVectorSingleDOF
 OFELI::Bar2DL2, [125](#)
 OFELI::Beam3DL2, [140](#)
 OFELI::DC1DL2, [179](#)
 OFELI::DC2DT3, [200](#)
 OFELI::DC2DT6, [218](#)
 OFELI::DC3DAT3, [237](#)
 OFELI::DC3DT4, [256](#)
 OFELI::EC2D1T3, [301](#)
 OFELI::Elas2DQ4, [325](#)
 OFELI::Elas2DT3, [345](#)
 OFELI::Elas3DH8, [361](#)
 OFELI::Elas3DT4, [376](#)
 OFELI::Equa_Electromagnetics, [399](#)
 OFELI::Equa_Fluid, [410](#)
 OFELI::Equa_Laplace, [423](#)
 OFELI::Equa_Solid, [439](#)
 OFELI::Equa_Therm, [454](#)
 OFELI::Equation, [466](#)
 OFELI::HelmholtzBT3, [500](#)
 OFELI::Laplace1DL2, [550](#)
 OFELI::Laplace1DL3, [563](#)
 OFELI::Laplace2DFVT, [576](#)
 OFELI::Laplace2DMHRT0, [590](#)
 OFELI::Laplace2DT3, [604](#)
 OFELI::NSP2DQ41, [709](#)
 OFELI::TINS2DT3B, [808](#)
 ElementSideVector
 OFELI::Bar2DL2, [125](#)
 OFELI::Beam3DL2, [140](#)
 OFELI::DC1DL2, [179](#)
 OFELI::DC2DT3, [201](#)
 OFELI::DC2DT6, [218](#)
 OFELI::DC3DAT3, [238](#)
 OFELI::DC3DT4, [258](#)
 OFELI::EC2D1T3, [302](#)
 OFELI::Elas2DQ4, [325](#)
 OFELI::Elas2DT3, [346](#)
 OFELI::Elas3DH8, [361](#)
 OFELI::Elas3DT4, [376](#)
 OFELI::Equa_Electromagnetics, [399](#)
 OFELI::Equa_Fluid, [412](#)
 OFELI::Equa_Laplace, [425](#)
 OFELI::Equa_Solid, [439](#)
 OFELI::Equa_Therm, [454](#)
 OFELI::Equation, [467](#)
 OFELI::HelmholtzBT3, [501](#)
 OFELI::Laplace1DL2, [550](#)
 OFELI::Laplace1DL3, [564](#)
 OFELI::Laplace2DFVT, [577](#)
 OFELI::Laplace2DMHRT0, [590](#)
 OFELI::Laplace2DT3, [604](#)
 OFELI::NSP2DQ41, [709](#)
 OFELI::TINS2DT3B, [808](#)

OFELI::TINS2DT3B, [808](#)
 ElementVector
 OFELI::Bar2DL2, [125](#)
 OFELI::Beam3DL2, [141](#)
 OFELI::DC1DL2, [180](#)
 OFELI::DC2DT3, [201](#)
 OFELI::DC2DT6, [218](#)
 OFELI::DC3DAT3, [238](#)
 OFELI::DC3DT4, [258](#)
 OFELI::EC2D1T3, [302](#)
 OFELI::Elas2DQ4, [326](#)
 OFELI::Elas2DT3, [346](#)
 OFELI::Elas3DH8, [361](#)
 OFELI::Elas3DT4, [376](#)
 OFELI::Equa_Electromagnetics, [399](#)
 OFELI::Equa_Fluid, [412](#)
 OFELI::Equa_Laplace, [425](#)
 OFELI::Equa_Solid, [439](#)
 OFELI::Equa_Therm, [454](#)
 OFELI::Equation, [467](#)
 OFELI::HelmholtzBT3, [501](#)
 OFELI::Laplace1DL2, [551](#)
 OFELI::Laplace1DL3, [564](#)
 OFELI::Laplace2DFVT, [577](#)
 OFELI::Laplace2DMHRT0, [590](#)
 OFELI::Laplace2DT3, [605](#)
 OFELI::NSP2DQ41, [709](#)
 OFELI::TINS2DT3B, [808](#)
 ElementsAreDOF
 OFELI::Mesh, [675](#)
 Ellipse, [392](#)
 OFELI::Ellipse, [393](#)
 EnthalpyToTemperature
 OFELI::PhaseChange, [721](#)
 EqDataType
 OFELI, [108](#)
 Equa_Fluid
 OFELI::Equa_Fluid, [409](#)
 Equa_Laplace
 OFELI::Equa_Laplace, [422](#)
 Equa_Solid
 OFELI::Equa_Solid, [436](#)
 Equa_Therm
 OFELI::Equa_Therm, [451](#)
 Equal
 Utilities, [83](#)
 Equation
 OFELI::Equation, [464](#), [465](#)
 Equation< T_, NEN_, NEE_, NSN_, NSE_ >, [461](#)
 Estimator, [474](#)
 Evaluate
 OFELI::FMM2D, [481](#)
 OFELI::FMM3D, [482](#)
 execute

- OFELI::FastMarching2D, [475](#)
- ExtendSpeed
 - OFELI::FMM2D, [481](#)
 - OFELI::FMM3D, [483](#)
 - OFELI::FMMSolver, [484](#)
- FE_2D_3N
 - OFELI, [110](#)
- FE_2D_4N
 - OFELI, [110](#)
- FE_2D_6N
 - OFELI, [110](#)
- FE_3D_4N
 - OFELI, [110](#)
- FE_3D_8N
 - OFELI, [110](#)
- FE_3D_AXI_3N
 - OFELI, [110](#)
- FIRST_ORDER_METHOD
 - OFELI::ICPG1D, [513](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [614](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [690](#)
 - OFELI::Muscl3DT, [695](#)
- FLUX
 - OFELI, [108](#)
- FORCE_WORD
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- FORWARD_EULER
 - OFELI, [109](#)
- FEShape, [475](#)
 - OFELI::FEShape, [477](#), [478](#)
- FEType
 - OFELI, [109](#)
- FMM2D, [480](#)
 - OFELI::FMM2D, [480](#)
- FMM3D, [481](#)
 - OFELI::FMM3D, [482](#)
- FMMSolver, [483](#)
 - OFELI::FMMSolver, [483](#)

- Factor
 - OFELI::BMatrix, [157](#)
 - OFELI::DMatrix, [271](#)
 - OFELI::DSMatrix, [287](#)
 - OFELI::LocalMatrix, [640](#)
 - OFELI::SkMatrix, [754](#)
 - OFELI::SkSMatrix, [768](#)
- FactorAndSolve
 - OFELI::BMatrix, [161](#)
 - OFELI::DMatrix, [277](#)
 - OFELI::DSMatrix, [293](#)
 - OFELI::LocalMatrix, [640](#)
 - OFELI::Matrix, [655](#), [656](#)
 - OFELI::SkMatrix, [759](#)
 - OFELI::SkSMatrix, [773](#)
 - OFELI::SpMatrix, [793](#)
 - OFELI::TrMatrix, [833](#)
- FastMarching2D, [474](#)
 - OFELI::FastMarching2D, [475](#)
- Figure, [478](#)
- Finite Element Mesh, [17](#)
 - DeformMesh, [23](#)
 - GRAPH_MEMORY, [21](#)
 - getMaxSideMeasure, [26](#)
 - getMaxSize, [25](#)
 - getMeanElementMeasure, [26](#)
 - getMeanSideMeasure, [26](#)
 - getMinElementMeasure, [25](#)
 - getMinSideMeasure, [25](#)
 - getMinSize, [25](#)
 - MAX_NB_CONT_LINES, [21](#)
 - MAX_NB_HOLES, [21](#)
 - MAX_NB_LINE_NODES, [21](#)
 - MAX_NB_LINES, [21](#)
 - MAX_NB_SUB_DOMAINS, [21](#)
 - MAX_NB_SUBMESHES, [21](#)
 - MeshActiveElements, [22](#)
 - MeshBoundaryNodes, [22](#)
 - MeshBoundarySides, [22](#)
 - MeshEdges, [23](#)
 - MeshElements, [21](#)
 - MeshNodeLoop, [22](#)
 - MeshNodes, [22](#)
 - MeshSideSet, [22](#)
 - MeshSides, [22](#)
 - MeshToMesh, [24](#)
 - NodeInElement, [27](#)
 - NodeInSide, [27](#)
 - operator-, [23](#)
 - operator&&, [23](#)
 - setBoundaryNodeCodes, [26](#)
 - setBoundarySideCodes, [27](#)
 - setElementCodes, [27](#)
 - setNodeCodes, [26](#)

- setSideCodes, [27](#)
- SideInElement, [29](#)
- TheEdge, [21](#)
- TheElement, [21](#)
- TheNode, [21](#)
- theNodeLabel, [23](#)
- TheSide, [21](#)
- Fluid Dynamics, [30](#)
- Forward
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [522](#)
 - OFELI::LCL1D, [616](#)
 - OFELI::LCL2DT, [622](#)
 - OFELI::LCL3DT, [628](#)
- Funct, [484](#)
 - OFELI::Funct, [485](#)
- GLOBAL_ARRAY
 - OFELI, [109](#)
- GMRES_SOLVER
 - OFELI, [110](#)
- GMRes
 - Solver, [52](#)
- GRAPH_MEMORY
 - Finite Element Mesh, [21](#)
- GS
 - Solver, [53](#)
- Gauss, [486](#)
- genMesh
 - OFELI::Domain, [280](#)
- General Purpose Equations, [31](#)
 - element_assembly, [32](#), [33](#)
 - side_assembly, [33](#), [34](#)
- generateMesh
 - OFELI::Domain, [281](#)
- get
 - OFELI::Domain, [280](#)
 - OFELI::IOField, [534](#)
 - OFELI::IPF, [540](#), [541](#)
 - OFELI::Mesh, [667](#), [668](#)
 - OFELI::Prescription, [733](#)
 - OFELI::SkMatrix, [755](#)
 - OFELI::SkSMatrix, [769](#)
 - OFELI::SpMatrix, [789](#)
- getActiveElement
 - OFELI::Mesh, [677](#)
- getAllEdges
 - OFELI::Mesh, [670](#)
- getAllSides
 - OFELI::Mesh, [669](#)
- getArray
 - OFELI::DMatrix, [273](#)
- getArraySize
 - OFELI::IPF, [540](#)

- getAuxFile
 - OFELI::IPF, [542](#)
- getAverage
 - OFELI::Vect, [852](#)
- getB1
 - OFELI::BiotSavart, [151](#)
- getB2
 - OFELI::BiotSavart, [151](#)
- getB3
 - OFELI::BiotSavart, [151](#)
- getBC
 - OFELI::IPF, [538](#)
- getBC1
 - OFELI::BiotSavart, [152](#)
- getBC2
 - OFELI::BiotSavart, [151](#)
- getBC3
 - OFELI::BiotSavart, [151](#)
- getBCFile
 - OFELI::IPF, [541](#)
- getBF
 - OFELI::IPF, [538](#)
- getBFFFile
 - OFELI::IPF, [542](#)
- getBamg
 - Utilities, [75](#)
- getBoundaryNodes
 - OFELI::Mesh, [669](#)
- getBoundarySides
 - OFELI::Mesh, [669](#)
- getCode
 - OFELI::Edge, [311](#)
 - OFELI::Grid, [493](#)
 - OFELI::Node, [700](#)
 - OFELI::Side, [743](#)
- getColHeight
 - OFELI::SkMatrix, [753](#)
 - OFELI::SkSMatrix, [767](#)
- getColInd
 - OFELI::DMatrix, [273](#)
 - OFELI::SpMatrix, [787](#)
- getComplex
 - OFELI::IPF, [540](#)
- getComplexPar
 - OFELI::IPF, [539](#)
- getCoord
 - OFELI::Node, [700](#)
- getCurl
 - OFELI::Vect, [851](#)
- getDOFType
 - OFELI::Vect, [846](#)
- getData
 - OFELI::IPF, [538](#)
- getDerivative

- OFELI::Tabulation, 797
- getDet
 - OFELI::FEShape, 478
 - OFELI::Hexa8, 510
 - OFELI::Line2, 631
 - OFELI::Line3, 634
 - OFELI::Penta6, 719
 - OFELI::Quad4, 736
 - OFELI::Tetra4, 800
 - OFELI::Triang3, 817
 - OFELI::Triang6S, 820
 - OFELI::triangle, 822, 824
- getDiag
 - OFELI::BMatrix, 157
 - OFELI::DMatrix, 273
 - OFELI::DSMatrix, 288
 - OFELI::Matrix, 651
 - OFELI::SkMatrix, 755
 - OFELI::SkSMatrix, 769
 - OFELI::SpMatrix, 789
 - OFELI::TrMatrix, 829
- getDivergence
 - OFELI::Vect, 852
- getDouble
 - OFELI::IPF, 540
- getDoublePar
 - OFELI::IPF, 539
- getEasymesh
 - Utilities, 76
- getEdgeLabel
 - OFELI::Mesh, 676
- getEigenValue
 - OFELI::AbsEqua, 116
 - OFELI::Bar2DL2, 131
 - OFELI::Beam3DL2, 148
 - OFELI::DC1DL2, 187
 - OFELI::DC2DT3, 208
 - OFELI::DC2DT6, 225
 - OFELI::DC3DAT3, 245
 - OFELI::DC3DT4, 265
 - OFELI::EC2D1T3, 309
 - OFELI::Elas2DQ4, 333
 - OFELI::Elas2DT3, 353
 - OFELI::Elas3DH8, 368
 - OFELI::Elas3DT4, 383
 - OFELI::Equa_Electromagnetics, 405
 - OFELI::Equa_Fluid, 419
 - OFELI::Equa_Laplace, 432
 - OFELI::Equa_Solid, 446
 - OFELI::Equa_Therm, 461
 - OFELI::Equation, 473
 - OFELI::HelmholtzBT3, 508
 - OFELI::Laplace1DL2, 558
 - OFELI::Laplace1DL3, 571
- OFELI::Laplace2DFVT, 584
- OFELI::Laplace2DMHRT0, 596
- OFELI::Laplace2DT3, 612
- OFELI::NSP2DQ41, 716
- OFELI::TINS2DT3B, 815
- getEigenVector
 - OFELI::AbsEqua, 116
 - OFELI::Bar2DL2, 131
 - OFELI::Beam3DL2, 148
 - OFELI::DC1DL2, 187
 - OFELI::DC2DT3, 208
 - OFELI::DC2DT6, 225
 - OFELI::DC3DAT3, 245
 - OFELI::DC3DT4, 265
 - OFELI::EC2D1T3, 309
 - OFELI::Eigen, 316
 - OFELI::Elas2DQ4, 333
 - OFELI::Elas2DT3, 353
 - OFELI::Elas3DH8, 368
 - OFELI::Elas3DT4, 383
 - OFELI::Equa_Electromagnetics, 405
 - OFELI::Equa_Fluid, 419
 - OFELI::Equa_Laplace, 432
 - OFELI::Equa_Solid, 446
 - OFELI::Equa_Therm, 461
 - OFELI::Equation, 473
 - OFELI::HelmholtzBT3, 508
 - OFELI::Laplace1DL2, 558
 - OFELI::Laplace1DL3, 571
 - OFELI::Laplace2DFVT, 584
 - OFELI::Laplace2DMHRT0, 596
 - OFELI::Laplace2DT3, 612
 - OFELI::NSP2DQ41, 716
 - OFELI::TINS2DT3B, 815
- getElementLabel
 - OFELI::Mesh, 676
- getElementNeighborElements
 - OFELI::Mesh, 670
- getGambit
 - Utilities, 76
- getGmsh
 - Utilities, 76
- getGradient
 - OFELI::Vect, 851
- getInit
 - OFELI::IPF, 538
- getInitFile
 - OFELI::IPF, 541
- getInnerProduct
 - OFELI::LocalMatrix, 641
- getIntPar
 - OFELI::IPF, 539
- getInteger
 - OFELI::IPF, 539

- getInternalEnergy
 - OFELI::ICPG1D, [514](#)
- getInterpolate
 - OFELI::Line2, [631](#)
- getList
 - OFELI::Mesh, [676](#)
- getLocal
 - OFELI::LocalVect, [644](#)
- getLocalPoint
 - OFELI::FEShape, [478](#)
 - OFELI::Hexa8, [510](#)
 - OFELI::Line2, [631](#)
 - OFELI::Line2H, [633](#)
 - OFELI::Line3, [634](#)
 - OFELI::Penta6, [719](#), [720](#)
 - OFELI::Quad4, [736](#)
 - OFELI::Tetra4, [800](#)
 - OFELI::Triang3, [817](#)
 - OFELI::Triang6S, [820](#)
 - OFELI::triangle, [822](#), [824](#)
- getMach
 - OFELI::ICPG1D, [515](#)
- getMatlab
 - Utilities, [77](#)
- getMaxSideMeasure
 - Finite Element Mesh, [26](#)
- getMaxSize
 - Finite Element Mesh, [25](#)
- getMaxTime
 - OFELI::IPF, [538](#)
- getMeanElementMeasure
 - Finite Element Mesh, [26](#)
- getMeanSideMeasure
 - Finite Element Mesh, [26](#)
- getMeasure
 - OFELI::Element, [390](#)
 - OFELI::Side, [744](#)
- getMesh
 - OFELI::AbsEqua, [115](#)
 - OFELI::Bar2DL2, [130](#)
 - OFELI::Beam3DL2, [146](#)
 - OFELI::DC1DL2, [185](#)
 - OFELI::DC2DT3, [206](#)
 - OFELI::DC2DT6, [223](#)
 - OFELI::DC3DAT3, [243](#)
 - OFELI::DC3DT4, [263](#)
 - OFELI::EC2D1T3, [307](#)
 - OFELI::Elas2DQ4, [331](#)
 - OFELI::Elas2DT3, [351](#)
 - OFELI::Elas3DH8, [367](#)
 - OFELI::Elas3DT4, [382](#)
 - OFELI::Equa_Electromagnetics, [404](#)
 - OFELI::Equa_Fluid, [417](#)
 - OFELI::Equa_Laplace, [430](#)
 - OFELI::Equa_Solid, [445](#)
 - OFELI::Equa_Therm, [459](#)
 - OFELI::Equation, [472](#)
 - OFELI::HelmholtzBT3, [506](#)
 - OFELI::Laplace1DL2, [556](#)
 - OFELI::Laplace1DL3, [569](#)
 - OFELI::Laplace2DFVT, [582](#)
 - OFELI::Laplace2DMHRT0, [595](#)
 - OFELI::Laplace2DT3, [610](#)
 - OFELI::NSP2DQ41, [714](#)
 - OFELI::TINS2DT3B, [813](#)
 - Utilities, [74](#)
- getMeshFile
 - OFELI::IPF, [541](#)
- getMinElementMeasure
 - Finite Element Mesh, [25](#)
- getMinSideMeasure
 - Finite Element Mesh, [25](#)
- getMinSize
 - Finite Element Mesh, [25](#)
- getMomentum
 - OFELI::ICPG1D, [514](#)
- getName
 - OFELI::Material, [648](#)
- getNbBoundarySides
 - OFELI::Mesh, [674](#)
- getNbInternalSides
 - OFELI::Mesh, [674](#)
- getNbIter
 - OFELI::IPF, [538](#)
- getNbNeigEl
 - OFELI::Node, [700](#)
- getNbNeigElements
 - OFELI::Element, [390](#)
- getNbSteps
 - OFELI::IPF, [538](#)
- getNeigEl
 - OFELI::Node, [700](#)
- getNeighborElement
 - OFELI::Element, [390](#)
 - OFELI::Side, [744](#)
- getNetgen
 - Utilities, [77](#)
- getNodeLabel
 - OFELI::Edge, [311](#)
 - OFELI::Mesh, [676](#)
- getNodeNeighborElements
 - OFELI::Mesh, [670](#)
- getNormal
 - OFELI::Side, [744](#)
- getNy
 - OFELI::Grid, [492](#)
- getNz
 - OFELI::Grid, [492](#)

- getOtherNeighborElement
 - OFELI::Side, [744](#)
- getOutput
 - OFELI::IPF, [537](#)
- getP
 - OFELI::ICPG2DT, [523](#)
- getPlot
 - OFELI::IPF, [538](#)
- getPlotFile
 - OFELI::IPF, [542](#)
- getPointDoublePar
 - OFELI::IPF, [539](#)
- getPrescriptionFile
 - OFELI::IPF, [542](#)
- getPrintView
 - OFELI::BMatrix, [163](#)
 - OFELI::DMatrix, [278](#)
 - OFELI::DSMatrix, [294](#)
 - OFELI::Matrix, [658](#)
 - OFELI::SkMatrix, [760](#)
 - OFELI::SkSMatrix, [774](#)
 - OFELI::SpMatrix, [794](#)
 - OFELI::TrMatrix, [835](#)
 - OFELI::Vect, [856](#)
- getProject
 - OFELI::IPF, [541](#)
- getR
 - OFELI::ICPG2DT, [523](#)
- getRefCoord
 - OFELI::Line2, [631](#)
- getRestartFile
 - OFELI::IPF, [541](#)
- getSCurl
 - OFELI::Vect, [852](#)
- getSF
 - OFELI::IPF, [538](#)
- getSFFile
 - OFELI::IPF, [542](#)
- getSave
 - OFELI::IPF, [537](#)
- getSaveFile
 - OFELI::IPF, [542](#)
- getSecondSideLabel
 - OFELI::Partition, [717](#)
- getSideLabel
 - OFELI::Mesh, [676](#)
- getSignedDistance
 - OFELI::Brick, [164](#), [165](#)
 - OFELI::Circle, [166](#)
 - OFELI::Ellipse, [393](#)
 - OFELI::Figure, [479](#)
 - OFELI::Polygon, [728](#)
 - OFELI::Rectangle, [738](#), [739](#)
 - OFELI::Sphere, [777](#)
- getSolution
 - OFELI::MeshAdapt, [680](#)
- getSolutionBB
 - OFELI::MeshAdapt, [680](#)
- getSolutionbb
 - OFELI::MeshAdapt, [680](#)
- getSoundSpeed
 - OFELI::ICPG1D, [515](#)
- getStresses
 - OFELI::Bar2DL2, [122](#)
- getString
 - OFELI::IPF, [539](#)
- getStringPar
 - OFELI::IPF, [539](#)
- getTetgen
 - Utilities, [77](#)
- getTime
 - OFELI::Timer, [801](#)
- getTimeStep
 - OFELI::IPF, [538](#)
- getTolerance
 - OFELI::IPF, [538](#)
- getTotalEnergy
 - OFELI::ICPG1D, [515](#)
- getTriangle
 - Utilities, [78](#)
- getUnitNormal
 - OFELI::Element, [390](#)
 - OFELI::Side, [744](#)
- getV
 - OFELI::ICPG2DT, [523](#)
- getValue
 - OFELI::Tabulation, [797](#), [798](#)
- getWNorm2
 - OFELI::Vect, [846](#)
- getXYZ
 - OFELI::Node, [700](#)
- Global Variables, [35](#)
 - Converged, [37](#)
 - InitPetsc, [37](#)
 - MaxNbIterations, [36](#)
 - NbTimeSteps, [36](#)
 - theEdge, [36](#)
 - theElement, [35](#)
 - theFinalTime, [37](#)
 - theliteration, [36](#)
 - theNode, [35](#)
 - theSide, [36](#)
 - theStep, [36](#)
 - theTime, [37](#)
 - theTimeStep, [36](#)
 - theTolerance, [37](#)
 - Verbosity, [36](#)
- Grad

- OFELI::DC2DT3, [197](#)
- OFELI::DC3DAT3, [234](#)
- OFELI::Quad4, [735](#)
- OFELI::Triang3, [817](#)
- OFELI::Triang6S, [819](#)
- Grid, [486](#)
 - OFELI::Grid, [489](#), [490](#)
- HEAT_FLUX
 - OFELI, [108](#)
- HEAT_SOURCE
 - OFELI, [108](#)
- HEUN
 - OFELI, [109](#)
- HLL_SOLVER
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- HLLC_SOLVER
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- Heat Transfer, [38](#)
- HelmholtzBT3, [495](#)
- Hexa8, [508](#)
- IDENT_PREC
 - OFELI, [111](#)
- IDENTITY
 - OFELI, [110](#)
- ILU_PREC
 - OFELI, [111](#)
- INCOMPRESSIBLE_NAVIER_STOKES
 - OFELI, [109](#)
- INITIAL_AUX_1
 - OFELI, [108](#)
- INITIAL_AUX_2
 - OFELI, [108](#)
- INITIAL_AUX_3
 - OFELI, [108](#)
- INITIAL_AUX_4
 - OFELI, [108](#)
- INITIAL_FIELD
 - OFELI, [108](#)
- INTERNAL_BOUNDARY
 - OFELI::Side, [742](#)
- INTERNAL_SIDE
 - OFELI::Side, [742](#)
- ICPG1D, [510](#)
 - OFELI::ICPG1D, [514](#)
- ICPG2DT, [518](#)
 - OFELI::ICPG2DT, [521](#)
- ICPG3DT, [525](#)
 - OFELI::ICPG3DT, [529](#), [530](#)
- IOField, [531](#)
 - OFELI::IOField, [532](#), [533](#)
- IPF, [535](#)
 - OFELI::IPF, [537](#)
- InitHeap
 - OFELI::FMM2D, [481](#)
 - OFELI::FMM3D, [482](#)
- InitPetsc
 - Global Variables, [37](#)
- InitialData
 - OFELI::UserData, [836](#)
- Initialize
 - OFELI::ICPG2DT, [524](#)
 - OFELI::LCL2DT, [622](#)
 - OFELI::Muscl2DT, [691](#)
- Input/Output, [39](#)
 - MAX_ARRAY_SIZE, [39](#)
 - MAX_NB_PAR, [39](#)
- insertBC
 - OFELI::Vect, [849](#), [850](#)
- insertCircle
 - OFELI::Domain, [281](#)
- insertLine
 - OFELI::Domain, [281](#)
- insertRequiredEdge
 - OFELI::Domain, [282](#)
- insertRequiredVertex
 - OFELI::Domain, [282](#)
- insertSubDomain
 - OFELI::Domain, [282](#)
- insertVertex
 - OFELI::Domain, [281](#)
- IntegND
 - OFELI::EC2D1T3, [299](#)
- Interface Problems, [40](#)
- Invert
 - OFELI::LocalMatrix, [641](#)
- isActive
 - OFELI::Grid, [495](#)
- isCloseTo

OFELI::Point, [724](#)
 isFactorized
 OFELI::BMatrix, [162](#)
 OFELI::DMatrix, [277](#)
 OFELI::DSMatrix, [293](#)
 OFELI::Matrix, [656](#)
 OFELI::SkMatrix, [759](#)
 OFELI::SkSMatrix, [773](#)
 OFELI::SpMatrix, [793](#)
 OFELI::TrMatrix, [834](#)
 isIn
 OFELI::Element, [391](#)
 isOnBoundary
 OFELI::Edge, [311](#)
 OFELI::Element, [390](#)
 OFELI::Node, [700](#)
 OFELI::Side, [744](#)
 Iter
 OFELI::Iter, [543](#)
 Iter < T_ >, [542](#)
 Iteration
 OFELI, [110](#)
 IterationLoop
 Solver, [48](#)

 Jacobi
 Solver, [53](#)
 JouleHeating
 OFELI::DC2DT3, [198](#)

 L_FORCE_WORD
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 LAPLACE
 OFELI, [109](#)
 LEAP_FROG
 OFELI, [109](#)
 LF_SOLVER
 OFELI::ICPG1D, [514](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)

 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 LINEARIZED_ELASTICITY
 OFELI, [109](#)
 LOAD
 OFELI, [108](#)
 LOCAL_ARRAY
 OFELI, [109](#)
 LORENTZ_FORCE
 OFELI, [108](#)
 LUMPED_CAPACITY
 OFELI, [108](#)
 LUMPED_MASS
 OFELI, [108](#)
 LCL1D, [612](#)
 LCL2DT, [617](#)
 OFELI::LCL2DT, [620](#)
 LCL3DT, [623](#)
 OFELI::LCL3DT, [626](#)
 LCapacity
 OFELI::DC1DL2, [174](#)
 OFELI::DC2DT3, [194](#)
 OFELI::DC3DAT3, [232](#)
 OFELI::DC3DT4, [251](#)
 LCapacityToLHS
 OFELI::DC1DL2, [173](#)
 OFELI::DC2DT3, [193](#)
 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [232](#)
 OFELI::DC3DT4, [251](#)
 OFELI::Equa_Therm, [451](#)
 LCapacityToRHS
 OFELI::DC1DL2, [173](#)
 OFELI::DC2DT3, [193](#)
 OFELI::DC2DT6, [215](#)
 OFELI::DC3DAT3, [232](#)
 OFELI::DC3DT4, [251](#)
 OFELI::Equa_Therm, [451](#)
 LHS
 OFELI::Laplace2DT3, [601](#)
 LHS1_Convection
 OFELI::NSP2DQ41, [707](#)
 LHS2_Convection
 OFELI::NSP2DQ41, [707](#)
 LMass
 OFELI::Bar2DL2, [121](#)
 OFELI::Elas2DT3, [340](#)
 LMassToLHS
 OFELI::Bar2DL2, [121](#)
 OFELI::Elas2DT3, [339](#)
 OFELI::Equa_Solid, [436](#)
 LMassToRHS
 OFELI::Bar2DL2, [122](#)
 OFELI::Elas2DT3, [339](#)

OFELI::Equa_Solid, [437](#)
 Laplace equation, [41](#)
 Laplace1DL2, [543](#)
 OFELI::Laplace1DL2, [547](#)
 Laplace1DL3, [558](#)
 OFELI::Laplace1DL3, [561](#)
 Laplace2DFVT, [571](#)
 OFELI::Laplace2DFVT, [574](#)
 Laplace2DMHRT0, [584](#)
 OFELI::Laplace2DMHRT0, [587](#)
 Laplace2DT3, [597](#)
 OFELI::Laplace2DT3, [600](#), [601](#)
 Limiter
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [614](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [690](#)
 OFELI::Muscl3DT, [695](#)
 Line2, [629](#)
 OFELI::Line2, [630](#)
 Line2H, [631](#)
 Line3, [633](#)
 LinearExchange
 OFELI::DC2DT3, [196](#)
 LocalMatrix
 OFELI::LocalMatrix, [636](#), [637](#)
 LocalMatrix< T_, NR_, NC_ >, [635](#)
 LocalNodeVector
 OFELI::Bar2DL2, [124](#)
 OFELI::Beam3DL2, [139](#)
 OFELI::DC1DL2, [178](#)
 OFELI::DC2DT3, [200](#)
 OFELI::DC2DT6, [217](#)
 OFELI::DC3DAT3, [237](#)
 OFELI::DC3DT4, [256](#)
 OFELI::EC2D1T3, [301](#)
 OFELI::Elas2DQ4, [324](#)
 OFELI::Elas2DT3, [345](#)
 OFELI::Elas3DH8, [360](#)
 OFELI::Elas3DT4, [375](#)
 OFELI::Equa_Electromagnetics, [398](#)
 OFELI::Equa_Fluid, [410](#)
 OFELI::Equa_Laplace, [423](#)
 OFELI::Equa_Solid, [438](#)
 OFELI::Equa_Therm, [453](#)
 OFELI::Equation, [466](#)
 OFELI::HelmholtzBT3, [500](#)
 OFELI::Laplace1DL2, [549](#)
 OFELI::Laplace1DL3, [563](#)

 OFELI::Laplace2DFVT, [576](#)
 OFELI::Laplace2DMHRT0, [589](#)
 OFELI::Laplace2DT3, [603](#)
 OFELI::NSP2DQ41, [708](#)
 OFELI::TINS2DT3B, [807](#)
 LocalVect
 OFELI::LocalVect, [643](#)
 LocalVect< T_, N_ >, [641](#)
 Localize
 OFELI::LocalMatrix, [637](#), [638](#)
 OFELI::LocalVect, [644](#)
 M_FORCE_WORD
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [614](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [690](#)
 OFELI::Muscl3DT, [695](#)
 MAGNETIC
 OFELI, [108](#)
 MASS
 OFELI, [108](#)
 MAX_LIMITER
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 MAX_SOLVER
 OFELI::ICPG1D, [514](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 MINMOD_LIMITER
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)

OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 MULTI_SLOPE_M_METHOD
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [614](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [690](#)
 OFELI::Muscl3DT, [695](#)
 MULTI_SLOPE_Q_METHOD
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [614](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [690](#)
 OFELI::Muscl3DT, [695](#)
 MAX_ARRAY_SIZE
 Input/Output, [39](#)
 MAX_NB_CONT_LINES
 Finite Element Mesh, [21](#)
 MAX_NB_EQUATIONS
 Solver, [48](#)
 MAX_NB_HOLES
 Finite Element Mesh, [21](#)
 MAX_NB_INPUT_FIELDS
 Solver, [48](#)
 MAX_NB_LINE_NODES
 Finite Element Mesh, [21](#)
 MAX_NB_LINES
 Finite Element Mesh, [21](#)
 MAX_NB_MESHES
 Solver, [48](#)
 MAX_NB_PAR
 Input/Output, [39](#)
 MAX_NB_SUB_DOMAINS
 Finite Element Mesh, [21](#)
 MAX_NB_SUBMESHES
 Finite Element Mesh, [21](#)
 Magnetic
 OFELI::EC2D1T3, [299](#)
 Mass
 OFELI::Bar2DL2, [121](#)

 OFELI::Elas2DT3, [340](#)
 MassToLHS
 OFELI::Bar2DL2, [122](#)
 OFELI::Elas2DQ4, [323](#)
 OFELI::Elas2DT3, [340](#)
 OFELI::Elas3DH8, [359](#)
 OFELI::Elas3DT4, [374](#)
 OFELI::Equa_Solid, [437](#)
 MassToRHS
 OFELI::Bar2DL2, [122](#)
 OFELI::Elas2DQ4, [323](#)
 OFELI::Elas2DT3, [340](#)
 OFELI::Elas3DH8, [359](#)
 OFELI::Elas3DT4, [374](#)
 OFELI::Equa_Solid, [437](#)
 Material, [646](#)
 OFELI::Material, [648](#)
 Matrix
 OFELI::Laplace1DL2, [548](#)
 OFELI::Laplace1DL3, [561](#)
 OFELI::Matrix, [651](#)
 Matrix< T_ >, [648](#)
 MatrixType
 OFELI, [110](#)
 MaxNbIterations
 Global Variables, [36](#)
 Media
 OFELI::Elas2DT3, [339](#)
 OFELI::Elas3DT4, [373](#)
 Mesh, [658](#)
 OFELI::Mesh, [664–666](#)
 MeshActiveElements
 Finite Element Mesh, [22](#)
 MeshAdapt, [677](#)
 OFELI::MeshAdapt, [679](#)
 MeshBoundaryNodes
 Finite Element Mesh, [22](#)
 MeshBoundarySides
 Finite Element Mesh, [22](#)
 MeshEdges
 Finite Element Mesh, [23](#)
 MeshElements
 Finite Element Mesh, [21](#)
 MeshNodeLoop
 Finite Element Mesh, [22](#)
 MeshNodes
 Finite Element Mesh, [22](#)
 MeshSideSet
 Finite Element Mesh, [22](#)
 MeshSides
 Finite Element Mesh, [22](#)
 MeshToMesh
 Finite Element Mesh, [24](#)
 Method

- OFELI::ICPG1D, [513](#)
- OFELI::ICPG2DT, [521](#)
- OFELI::ICPG3DT, [528](#)
- OFELI::LCL1D, [614](#)
- OFELI::LCL2DT, [620](#)
- OFELI::LCL3DT, [626](#)
- OFELI::Muscl, [683](#)
- OFELI::Muscl1D, [687](#)
- OFELI::Muscl2DT, [690](#)
- OFELI::Muscl3DT, [695](#)
- Mult
 - OFELI::DMatrix, [270](#)
 - OFELI::LocalMatrix, [640](#)
 - OFELI::SkMatrix, [752](#)
 - OFELI::SkSMatrix, [767](#)
 - OFELI::SpMatrix, [786](#)
- MultAdd
 - OFELI::DMatrix, [270](#)
 - OFELI::DSMatrix, [287](#)
 - OFELI::LocalMatrix, [638](#)
 - OFELI::SkMatrix, [752](#)
 - OFELI::SkSMatrix, [766](#), [767](#)
 - OFELI::SpMatrix, [786](#)
 - OFELI::Vect, [852](#)
- MultAddScal
 - OFELI::LocalMatrix, [640](#)
- Muscl, [681](#)
- Muscl1D, [685](#)
- Muscl2DT, [689](#)
- Muscl3DT, [692](#)
- NEWMARK
 - OFELI, [109](#)
- NSP2DQ41, [702](#)
 - OFELI::NSP2DQ41, [706](#), [707](#)
- NbTimeSteps
 - Global Variables, [36](#)
- Node, [696](#)
 - OFELI::Node, [698](#)
- NodeInElement
 - Finite Element Mesh, [27](#)
- NodeInSide
 - Finite Element Mesh, [27](#)
- NodeList, [701](#)
- NodesAreDOF
 - OFELI::Mesh, [675](#)
- Normalize
 - OFELI::Point, [723](#)
- Nrm2
 - Utilities, [82](#)
- NumberEquations
 - OFELI::Mesh, [669](#)
- OFELI
 - AB2, [109](#)
 - ADAMS_BASHFORTH, [109](#)
 - AUX_INPUT_FIELD_1, [108](#)
 - AUX_INPUT_FIELD_2, [109](#)
 - AUX_INPUT_FIELD_3, [109](#)
 - AUX_INPUT_FIELD_4, [109](#)
 - BACKWARD_EULER, [109](#)
 - BICG_SOLVER, [110](#)
 - BICG_STAB_SOLVER, [110](#)
 - BODY_FORCE, [108](#)
 - BOUNDARY_CONDITION, [108](#)
 - BOUNDARY_FORCE, [108](#)
 - BOUNDARY_TRACTION, [108](#)
 - BUOYANCY, [108](#)
 - CAPACITY, [108](#)
 - CG_SOLVER, [110](#)
 - CGS_SOLVER, [110](#)
 - CONTACT, [108](#)
 - CONTACT_BC, [111](#)
 - CONVECTION, [108](#)
 - CRANK_NICOLSON, [109](#)
 - DEVIATORIC, [108](#)
 - DIAG_PREC, [111](#)
 - DIAGONAL, [110](#)
 - DIFFUSION, [108](#)
 - DIFFUSION_CONVECTION, [109](#)
 - DILATATION, [108](#)
 - DIRECT_SOLVER, [110](#)
 - DISPLACEMENT_FIELD, [109](#)
 - ELECTRIC, [108](#)
 - FE_2D_3N, [110](#)
 - FE_2D_4N, [110](#)
 - FE_2D_6N, [110](#)
 - FE_3D_4N, [110](#)
 - FE_3D_8N, [110](#)
 - FE_3D_AXI_3N, [110](#)
 - FLUX, [108](#)
 - FORWARD_EULER, [109](#)
 - GLOBAL_ARRAY, [109](#)
 - GMRES_SOLVER, [110](#)
 - HEAT_FLUX, [108](#)
 - HEAT_SOURCE, [108](#)
 - HEUN, [109](#)
 - IDENT_PREC, [111](#)
 - IDENTITY, [110](#)
 - ILU_PREC, [111](#)
 - INCOMPRESSIBLE_NAVIER_STOKES, [109](#)
 - INITIAL_AUX_1, [108](#)
 - INITIAL_AUX_2, [108](#)
 - INITIAL_AUX_3, [108](#)
 - INITIAL_AUX_4, [108](#)
 - INITIAL_FIELD, [108](#)
 - LAPLACE, [109](#)
 - LEAP_FROG, [109](#)

LINEARIZED_ELASTICITY, 109
 LOAD, 108
 LOCAL_ARRAY, 109
 LORENTZ_FORCE, 108
 LUMPED_CAPACITY, 108
 LUMPED_MASS, 108
 MAGNETIC, 108
 MASS, 108
 NEWMARK, 109
 OPTIMIZATION, 110
 PERIODIC_A, 111
 PERIODIC_B, 111
 PLANAR_TRUSS, 109
 POINT_FORCE, 108
 QMR_SOLVER, 110
 RK4, 109
 RUNGE_KUTTA, 109
 SKYLINE, 110
 SLIP, 111
 SOLUTION, 108
 SOURCE, 108
 SPARSE, 110
 SPATIAL_BEAM, 109
 SSOR_PREC, 111
 STATIONARY, 109
 STEADY_STATE, 110
 STIFFNESS, 108
 SYMMETRIC, 110
 TEMPERATURE_FIELD, 109
 THERMAL_PHASE_CHANGE, 109
 TRACTION, 108
 TRANSIENT, 110
 TRIDIAGONAL, 110
 UNSYMMETRIC, 110
 VELOCITY_FIELD, 109
 VISCOSITY, 108
 OFELI::ICPG1D
 FIRST_ORDER_METHOD, 513
 FORCE_WORD, 514
 HLL_SOLVER, 514
 HLLC_SOLVER, 514
 L_FORCE_WORD, 513
 LF_SOLVER, 514
 M_FORCE_WORD, 513
 MAX_LIMITER, 513
 MAX_SOLVER, 514
 MINMOD_LIMITER, 513
 MULTI_SLOPE_M_METHOD, 513
 MULTI_SLOPE_Q_METHOD, 513
 ROE_SOLVER, 514
 RUSANOV_SOLVER, 514
 SUPERBEE_LIMITER, 513
 VANALBADA_LIMITER, 513
 VANLEER_LIMITER, 513

 VFROE_SOLVER, 514
 OFELI::ICPG2DT
 FIRST_ORDER_METHOD, 521
 FORCE_WORD, 521
 HLL_SOLVER, 521
 HLLC_SOLVER, 521
 L_FORCE_WORD, 521
 LF_SOLVER, 521
 M_FORCE_WORD, 521
 MAX_LIMITER, 521
 MAX_SOLVER, 521
 MINMOD_LIMITER, 521
 MULTI_SLOPE_M_METHOD, 521
 MULTI_SLOPE_Q_METHOD, 521
 ROE_SOLVER, 521
 RUSANOV_SOLVER, 521
 SUPERBEE_LIMITER, 521
 VANALBADA_LIMITER, 521
 VANLEER_LIMITER, 521
 VFROE_SOLVER, 521
 OFELI::ICPG3DT
 FIRST_ORDER_METHOD, 528
 FORCE_WORD, 528
 HLL_SOLVER, 528
 HLLC_SOLVER, 528
 L_FORCE_WORD, 528
 LF_SOLVER, 528
 M_FORCE_WORD, 528
 MAX_LIMITER, 528
 MAX_SOLVER, 528
 MINMOD_LIMITER, 528
 MULTI_SLOPE_M_METHOD, 528
 MULTI_SLOPE_Q_METHOD, 528
 ROE_SOLVER, 528
 RUSANOV_SOLVER, 528
 SUPERBEE_LIMITER, 528
 VANALBADA_LIMITER, 528
 VANLEER_LIMITER, 528
 VFROE_SOLVER, 528
 OFELI::LCL1D
 FIRST_ORDER_METHOD, 614
 FORCE_WORD, 615
 HLL_SOLVER, 615
 HLLC_SOLVER, 615
 L_FORCE_WORD, 615
 LF_SOLVER, 615
 M_FORCE_WORD, 614
 MAX_LIMITER, 615
 MAX_SOLVER, 615
 MINMOD_LIMITER, 615
 MULTI_SLOPE_M_METHOD, 614
 MULTI_SLOPE_Q_METHOD, 614
 ROE_SOLVER, 615
 RUSANOV_SOLVER, 615

SUPERBEE_LIMITER, 615
 VANALBADA_LIMITER, 615
 VANLEER_LIMITER, 615
 VFROE_SOLVER, 615
 OFELI::LCL2DT
 FIRST_ORDER_METHOD, 620
 FORCE_WORD, 620
 HLL_SOLVER, 620
 HLLC_SOLVER, 620
 L_FORCE_WORD, 620
 LF_SOLVER, 620
 M_FORCE_WORD, 620
 MAX_LIMITER, 620
 MAX_SOLVER, 620
 MINMOD_LIMITER, 620
 MULTI_SLOPE_M_METHOD, 620
 MULTI_SLOPE_Q_METHOD, 620
 ROE_SOLVER, 620
 RUSANOV_SOLVER, 620
 SUPERBEE_LIMITER, 620
 VANALBADA_LIMITER, 620
 VANLEER_LIMITER, 620
 VFROE_SOLVER, 620
 OFELI::LCL3DT
 FIRST_ORDER_METHOD, 626
 FORCE_WORD, 626
 HLL_SOLVER, 626
 HLLC_SOLVER, 626
 L_FORCE_WORD, 626
 LF_SOLVER, 626
 M_FORCE_WORD, 626
 MAX_LIMITER, 626
 MAX_SOLVER, 626
 MINMOD_LIMITER, 626
 MULTI_SLOPE_M_METHOD, 626
 MULTI_SLOPE_Q_METHOD, 626
 ROE_SOLVER, 626
 RUSANOV_SOLVER, 626
 SUPERBEE_LIMITER, 626
 VANALBADA_LIMITER, 626
 VANLEER_LIMITER, 626
 VFROE_SOLVER, 626
 OFELI::Muscl
 FIRST_ORDER_METHOD, 683
 FORCE_WORD, 683
 HLL_SOLVER, 683
 HLLC_SOLVER, 683
 L_FORCE_WORD, 683
 LF_SOLVER, 683
 M_FORCE_WORD, 683
 MAX_LIMITER, 683
 MAX_SOLVER, 683
 MINMOD_LIMITER, 683
 MULTI_SLOPE_M_METHOD, 683

MULTI_SLOPE_Q_METHOD, 683
 ROE_SOLVER, 683
 RUSANOV_SOLVER, 683
 SUPERBEE_LIMITER, 683
 VANALBADA_LIMITER, 683
 VANLEER_LIMITER, 683
 VFROE_SOLVER, 683
 OFELI::Muscl1D
 FIRST_ORDER_METHOD, 687
 FORCE_WORD, 687
 HLL_SOLVER, 687
 HLLC_SOLVER, 687
 L_FORCE_WORD, 687
 LF_SOLVER, 687
 M_FORCE_WORD, 687
 MAX_LIMITER, 687
 MAX_SOLVER, 687
 MINMOD_LIMITER, 687
 MULTI_SLOPE_M_METHOD, 687
 MULTI_SLOPE_Q_METHOD, 687
 ROE_SOLVER, 687
 RUSANOV_SOLVER, 687
 SUPERBEE_LIMITER, 687
 VANALBADA_LIMITER, 687
 VANLEER_LIMITER, 687
 VFROE_SOLVER, 687
 OFELI::Muscl2DT
 FIRST_ORDER_METHOD, 690
 FORCE_WORD, 691
 HLL_SOLVER, 691
 HLLC_SOLVER, 691
 L_FORCE_WORD, 691
 LF_SOLVER, 691
 M_FORCE_WORD, 690
 MAX_LIMITER, 691
 MAX_SOLVER, 691
 MINMOD_LIMITER, 691
 MULTI_SLOPE_M_METHOD, 690
 MULTI_SLOPE_Q_METHOD, 690
 ROE_SOLVER, 691
 RUSANOV_SOLVER, 691
 SUPERBEE_LIMITER, 691
 VANALBADA_LIMITER, 691
 VANLEER_LIMITER, 691
 VFROE_SOLVER, 691
 OFELI::Muscl3DT
 FIRST_ORDER_METHOD, 695
 FORCE_WORD, 695
 HLL_SOLVER, 695
 HLLC_SOLVER, 695
 L_FORCE_WORD, 695
 LF_SOLVER, 695
 M_FORCE_WORD, 695
 MAX_LIMITER, 695

- MAX_SOLVER, 695
- MINMOD_LIMITER, 695
- MULTI_SLOPE_M_METHOD, 695
- MULTI_SLOPE_Q_METHOD, 695
- ROE_SOLVER, 695
- RUSANOV_SOLVER, 695
- SUPERBEE_LIMITER, 695
- VANALBADA_LIMITER, 695
- VANLEER_LIMITER, 695
- VFROE_SOLVER, 695
- OFELI::Side
 - EXTERNAL_BOUNDARY, 742
 - INTERNAL_BOUNDARY, 742
 - INTERNAL_SIDE, 742
- OPTIMIZATION
 - OFELI, 110
- OFELI, 42, 89
 - A, 111
 - AnalysisType, 110
 - ArrayType, 109
 - b, 111
 - BCType, 111
 - EqDataType, 108
 - FEType, 109
 - Iteration, 110
 - MatrixType, 110
 - PDE, 109
 - PDE_Terms, 108
 - Preconditioner, 110
 - Prev, 111
 - TimeScheme, 109
- OFELI::AbsEqua
 - getEigenValue, 116
 - getEigenVector, 116
 - getMesh, 115
 - setSolver, 115
 - solveEigenProblem, 116
- OFELI::BMatrix
 - Assembly, 157, 158
 - Axpy, 156
 - BMatrix, 155
 - Factor, 157
 - FactorAndSolve, 161
 - getDiag, 157
 - getPrintView, 163
 - isFactorized, 162
 - operator*=, 157
 - operator(), 156, 162
 - operator+=, 157, 162
 - operator-=, 162
 - operator=, 156
 - Prescribe, 158–160
 - PrescribeSide, 161
 - setDiagonal, 157
 - setPrintView, 163
 - setSize, 156
 - Solve, 157, 161
- OFELI::Bar2DL2
 - AxbAssembly, 130
 - Bar2DL2, 121
 - BodyRHS, 122
 - buildEigen, 123
 - DGELEMENTAssembly, 128
 - DiagBC, 124
 - ElementAssembly, 126, 127
 - ElementNodeCoordinates, 126
 - ElementNodeVector, 124
 - ElementNodeVectorSingleDOF, 125
 - ElementSideVector, 125
 - ElementVector, 125
 - getEigenValue, 131
 - getEigenVector, 131
 - getMesh, 130
 - getStresses, 122
 - LMass, 121
 - LMassToLHS, 121
 - LMassToRHS, 122
 - LocalNodeVector, 124
 - Mass, 121
 - MassToLHS, 122
 - MassToRHS, 122
 - setMaterialProperty, 130
 - setSolver, 130
 - SideAssembly, 129
 - SideNodeCoordinates, 126
 - SideVector, 126
 - solveEigenProblem, 131
 - Stiffness, 122
 - updateBC, 123
- OFELI::Beam3DL2
 - AxbAssembly, 145, 146
 - Beam3DL2, 137, 138
 - buildEigen, 138
 - DGELEMENTAssembly, 143, 144
 - DiagBC, 139
 - ElementAssembly, 142, 143
 - ElementNodeCoordinates, 142
 - ElementNodeVector, 140
 - ElementNodeVectorSingleDOF, 140
 - ElementSideVector, 140
 - ElementVector, 141
 - getEigenValue, 148
 - getEigenVector, 148
 - getMesh, 146
 - LocalNodeVector, 139
 - setMaterialProperty, 146
 - setSolver, 147
 - SideAssembly, 144, 145

- SideNodeCoordinates, 142
- SideVector, 141
- solveEigenProblem, 147
- updateBC, 139
- OFELI::BiotSavart
 - BiotSavart, 149
 - getB1, 151
 - getB2, 151
 - getB3, 151
 - getBC1, 152
 - getBC2, 151
 - getBC3, 151
 - run, 152
 - setBoundary, 150
 - setCurrentDensity, 150
 - setMagneticInduction, 150
 - setPermeability, 150
- OFELI::Brick
 - Brick, 164
 - dLine, 165
 - getSignedDistance, 164, 165
 - operator+=, 164
 - setBoundingBox, 164
- OFELI::Circle
 - Circle, 166
 - dLine, 168
 - getSignedDistance, 166
 - operator+=, 166
- OFELI::DC1DL2
 - AxbAssembly, 184, 185
 - BodyRHS, 175, 176
 - BoundaryRHS, 176
 - buildEigen, 177
 - Capacity, 174
 - CapacityToLHS, 174
 - CapacityToRHS, 174
 - Convection, 175
 - ConvectionToRHS, 175
 - DC1DL2, 173
 - DGElementAssembly, 182, 183
 - DiagBC, 178
 - Diffusion, 174
 - DiffusionToRHS, 174
 - ElementAssembly, 181, 182
 - ElementNodeCoordinates, 181
 - ElementNodeVector, 179
 - ElementNodeVectorSingleDOF, 179
 - ElementSideVector, 179
 - ElementVector, 180
 - getEigenValue, 187
 - getEigenVector, 187
 - getMesh, 185
 - LCapacity, 174
 - LCapacityToLHS, 173
 - LCapacityToRHS, 173
 - LocalNodeVector, 178
 - run, 177
 - runTransient, 177
 - setInput, 176
 - setMaterialProperty, 185
 - setSolver, 186
 - setStab, 177
 - SideAssembly, 183, 184
 - SideNodeCoordinates, 181
 - SideVector, 180
 - solveEigenProblem, 186
 - updateBC, 178
- OFELI::DC2DT3
 - AxbAssembly, 205, 206
 - BodyRHS, 196
 - BoundaryRHS, 196, 197
 - buildEigen, 198, 199
 - Capacity, 194
 - CapacityToLHS, 194
 - CapacityToRHS, 194
 - Convection, 195
 - ConvectionToRHS, 195
 - DC2DT3, 192, 193
 - DGElementAssembly, 203, 204
 - DiagBC, 199
 - Diffusion, 194
 - DiffusionToRHS, 195
 - ElementAssembly, 202, 203
 - ElementNodeCoordinates, 202
 - ElementNodeVector, 200
 - ElementNodeVectorSingleDOF, 200
 - ElementSideVector, 201
 - ElementVector, 201
 - getEigenValue, 208
 - getEigenVector, 208
 - getMesh, 206
 - Grad, 197
 - JouleHeating, 198
 - LCapacity, 194
 - LCapacityToLHS, 193
 - LCapacityToRHS, 193
 - LinearExchange, 196
 - LocalNodeVector, 200
 - Periodic, 197
 - run, 198
 - runSteadyState, 198
 - runTransient, 198
 - setInput, 197
 - setMaterialProperty, 206
 - setSolver, 207
 - setStab, 198
 - SideAssembly, 204, 205
 - SideNodeCoordinates, 202

- SideVector, 201
- solveEigenProblem, 207
- updateBC, 199
- OFELI::DC2DT6
 - AxbAssembly, 223
 - BodyRHS, 215
 - BoundaryRHS, 215
 - build, 216
 - buildEigen, 216
 - CapacityToLHS, 215
 - CapacityToRHS, 215
 - Convection, 214
 - DC2DT6, 212, 213
 - DGElementAssembly, 221, 222
 - DiagBC, 217
 - Diffusion, 214
 - ElementAssembly, 220, 221
 - ElementNodeCoordinates, 219
 - ElementNodeVector, 217, 218
 - ElementNodeVectorSingleDOF, 218
 - ElementSideVector, 218
 - ElementVector, 218
 - getEigenValue, 225
 - getEigenVector, 225
 - getMesh, 223
 - LCapacityToLHS, 215
 - LCapacityToRHS, 215
 - LocalNodeVector, 217
 - run, 216
 - runTransient, 216
 - setMaterialProperty, 223
 - setSolver, 224
 - setStab, 215
 - SideAssembly, 222, 223
 - SideNodeCoordinates, 219
 - SideVector, 219
 - solveEigenProblem, 225
 - updateBC, 216, 217
- OFELI::DC3DAT3
 - AxbAssembly, 242, 243
 - BodyRHS, 233, 234
 - BoundaryRHS, 234
 - buildEigen, 234, 236
 - Capacity, 233
 - CapacityToLHS, 232
 - CapacityToRHS, 233
 - DC3DAT3, 230, 232
 - DGElementAssembly, 240, 241
 - DiagBC, 236
 - Diffusion, 233
 - DiffusionToRHS, 233
 - ElementAssembly, 239, 240
 - ElementNodeCoordinates, 239
 - ElementNodeVector, 237
 - ElementNodeVectorSingleDOF, 237
 - ElementSideVector, 238
 - ElementVector, 238
 - getEigenValue, 245
 - getEigenVector, 245
 - getMesh, 243
 - Grad, 234
 - LCapacity, 232
 - LCapacityToLHS, 232
 - LCapacityToRHS, 232
 - LocalNodeVector, 237
 - run, 234
 - runTransient, 234
 - setMaterialProperty, 243
 - setSolver, 244
 - setStab, 234
 - SideAssembly, 241, 242
 - SideNodeCoordinates, 239
 - SideVector, 238
 - solveEigenProblem, 244
 - updateBC, 236
- OFELI::DC3DT4
 - AxbAssembly, 262, 263
 - BodyRHS, 253
 - BoundaryRHS, 254
 - buildEigen, 255
 - Capacity, 252
 - CapacityToLHS, 252
 - CapacityToRHS, 252
 - Convection, 253
 - DC3DT4, 250, 251
 - DGElementAssembly, 260, 261
 - DiagBC, 255
 - Diffusion, 252
 - DiffusionToRHS, 252
 - ElementAssembly, 259, 260
 - ElementNodeCoordinates, 259
 - ElementNodeVector, 256
 - ElementNodeVectorSingleDOF, 256
 - ElementSideVector, 258
 - ElementVector, 258
 - getEigenValue, 265
 - getEigenVector, 265
 - getMesh, 263
 - LCapacity, 251
 - LCapacityToLHS, 251
 - LCapacityToRHS, 251
 - LocalNodeVector, 256
 - Periodic, 254
 - RHS_Convection, 253
 - run, 255
 - runTransient, 254
 - setMaterialProperty, 263
 - setSolver, 264

- setStab, 254
- SideAssembly, 261, 262
- SideNodeCoordinates, 259
- SideVector, 258
- solveEigenProblem, 264
- updateBC, 255
- OFELI::DMatrix
 - add, 270
 - Assembly, 273, 274
 - Axpy, 271
 - DMatrix, 268, 269
 - Factor, 271
 - FactorAndSolve, 277
 - getArray, 273
 - getColInd, 273
 - getDiag, 273
 - getPrintView, 278
 - isFactorized, 277
 - Mult, 270
 - MultAdd, 270
 - operator*=, 272
 - operator(), 271, 277
 - operator+=, 272, 278
 - operator=, 272, 278
 - operator=, 272
 - Prescribe, 274–276
 - PrescribeSide, 276
 - set, 269
 - setColumn, 270
 - setDiagonal, 273
 - setPrintView, 273
 - setRow, 269
 - setSize, 269
 - Solve, 271, 272
 - TMult, 270
- OFELI::DSMatrix
 - add, 286
 - Assembly, 288, 290
 - Axpy, 287
 - DSMatrix, 285, 286
 - Factor, 287
 - FactorAndSolve, 293
 - getDiag, 288
 - getPrintView, 294
 - isFactorized, 293
 - MultAdd, 287
 - operator*=, 294
 - operator(), 286, 293, 294
 - operator+=, 294
 - operator=, 294
 - Prescribe, 290–292
 - PrescribeSide, 293
 - set, 286
 - setDiagonal, 288

- setPrintView, 288
- setSize, 286
- Solve, 288
- TMult, 287
- OFELI::Domain
 - Domain, 280
 - genMesh, 280
 - generateMesh, 281
 - get, 280
 - insertCircle, 281
 - insertLine, 281
 - insertRequiredEdge, 282
 - insertRequiredVertex, 282
 - insertSubDomain, 282
 - insertVertex, 281
 - operator*=, 281
- OFELI::EC2D1T3
 - AxbAssembly, 306, 307
 - DGELEMENTAssembly, 304, 305
 - DiagBC, 300
 - EC2D1T3, 298, 299
 - Electric, 299
 - ElementAssembly, 303, 304
 - ElementNodeCoordinates, 303
 - ElementNodeVector, 301
 - ElementNodeVectorSingleDOF, 301
 - ElementSideVector, 302
 - ElementVector, 302
 - getEigenValue, 309
 - getEigenVector, 309
 - getMesh, 307
 - IntegND, 299
 - LocalNodeVector, 301
 - Magnetic, 299
 - setMaterialProperty, 307
 - setSolver, 308
 - SideAssembly, 305, 306
 - SideNodeCoordinates, 303
 - SideVector, 302
 - solveEigenProblem, 308
 - updateBC, 300
- OFELI::Edge
 - DOF, 310
 - Edge, 310
 - getCode, 311
 - getNodeLabel, 311
 - isOnBoundary, 311
 - operator(), 311
 - setCode, 311
 - setDOF, 311
- OFELI::EdgeList
 - selectCode, 312
 - unselectCode, 312
- OFELI::Eigen

- checkSturm, 315
- Eigen, 313, 314
- getEigenVector, 316
- runSubSpace, 315
- setMatrix, 314, 315
- setSubspaceDimension, 315
- setTolerance, 315
- OFELI::Elas2DQ4
 - AxbAssembly, 330, 331
 - BodyRHS, 322
 - BoundaryRHS, 322
 - buildEigen, 323
 - DGElementAssembly, 328, 329
 - DiagBC, 324
 - DilatationToRHS, 322
 - Elas2DQ4, 321
 - ElementAssembly, 327, 328
 - ElementNodeCoordinates, 327
 - ElementNodeVector, 325
 - ElementNodeVectorSingleDOF, 325
 - ElementSideVector, 325
 - ElementVector, 326
 - getEigenValue, 333
 - getEigenVector, 333
 - getMesh, 331
 - LocalNodeVector, 324
 - MassToLHS, 323
 - MassToRHS, 323
 - PlaneStrain, 321
 - PlaneStress, 321
 - setMaterialProperty, 331
 - setSolver, 332
 - SideAssembly, 329, 330
 - SideNodeCoordinates, 327
 - SideVector, 326
 - SignoriniContact, 322
 - solveEigenProblem, 332
 - Strain, 322
 - Stress, 323
 - updateBC, 324
- OFELI::Elas2DT3
 - AxbAssembly, 350, 351
 - BodyRHS, 341
 - BoundaryRHS, 341
 - buildEigen, 343, 344
 - ContactPressure, 343
 - DGElementAssembly, 348, 349
 - Deviator, 340
 - DeviatorToRHS, 340
 - DiagBC, 344
 - Dilatation, 340
 - DilatationToRHS, 341
 - Elas2DT3, 338, 339
 - ElementAssembly, 347, 348
 - ElementNodeCoordinates, 347
 - ElementNodeVector, 345
 - ElementNodeVectorSingleDOF, 345
 - ElementSideVector, 346
 - ElementVector, 346
 - getEigenValue, 353
 - getEigenVector, 353
 - getMesh, 351
 - LMass, 340
 - LMassToLHS, 339
 - LMassToRHS, 339
 - LocalNodeVector, 345
 - Mass, 340
 - MassToLHS, 340
 - MassToRHS, 340
 - Media, 339
 - Periodic, 343
 - Reaction, 343
 - setMaterialProperty, 351
 - setSolver, 352
 - SideAssembly, 349, 350
 - SideNodeCoordinates, 347
 - SideVector, 346
 - SignoriniContact, 341
 - solveEigenProblem, 352
 - Strain, 343
 - Stress, 343
 - updateBC, 344
- OFELI::Elas3DH8
 - AxbAssembly, 367
 - BodyRHS, 357, 359
 - BoundaryRHS, 357
 - buildEigen, 359
 - DGElementAssembly, 365
 - DiagBC, 360
 - Elas3DH8, 357
 - ElementAssembly, 362, 364
 - ElementNodeCoordinates, 362
 - ElementNodeVector, 360, 361
 - ElementNodeVectorSingleDOF, 361
 - ElementSideVector, 361
 - ElementVector, 361
 - getEigenValue, 368
 - getEigenVector, 368
 - getMesh, 367
 - LocalNodeVector, 360
 - MassToLHS, 359
 - MassToRHS, 359
 - setMaterialProperty, 367
 - setSolver, 367
 - SideAssembly, 366
 - SideNodeCoordinates, 362
 - SideVector, 362
 - solveEigenProblem, 368

- updateBC, [359](#), [360](#)
- OFELI::Elas3DT4
 - AxbAssembly, [382](#)
 - BodyRHS, [373](#)
 - BoundaryRHS, [374](#)
 - buildEigen, [374](#)
 - DGELEMENTAssembly, [380](#)
 - DiagBC, [375](#)
 - ElementAssembly, [377](#), [379](#)
 - ElementNodeCoordinates, [377](#)
 - ElementNodeVector, [375](#), [376](#)
 - ElementNodeVectorSingleDOF, [376](#)
 - ElementSideVector, [376](#)
 - ElementVector, [376](#)
 - getEigenValue, [383](#)
 - getEigenVector, [383](#)
 - getMesh, [382](#)
 - LocalNodeVector, [375](#)
 - MassToLHS, [374](#)
 - MassToRHS, [374](#)
 - Media, [373](#)
 - setMaterialProperty, [382](#)
 - setSolver, [382](#)
 - SideAssembly, [381](#)
 - SideNodeCoordinates, [377](#)
 - SideVector, [377](#)
 - solveEigenProblem, [383](#)
 - updateBC, [374](#), [375](#)
- OFELI::Element
 - Add, [388](#), [389](#)
 - Contains, [389](#)
 - Element, [386](#), [387](#)
 - getMeasure, [390](#)
 - getNbNeigElements, [390](#)
 - getNeighborElement, [390](#)
 - getUnitNormal, [390](#)
 - IsIn, [391](#)
 - isOnBoundary, [390](#)
 - operator(), [389](#), [390](#)
 - Replace, [388](#)
 - set, [389](#)
 - setChild, [391](#)
 - setCode, [387](#), [389](#)
 - setDOF, [389](#)
 - setLabel, [387](#)
 - setSide, [390](#)
- OFELI::ElementList
 - selectLevel, [392](#)
 - unselectCode, [392](#)
- OFELI::Ellipse
 - dLine, [394](#)
 - Ellipse, [393](#)
 - getSignedDistance, [393](#)
 - operator+=, [393](#)

- OFELI::Equa_Electromagnetics
 - AxbAssembly, [404](#)
 - DGELEMENTAssembly, [402](#)
 - DiagBC, [398](#)
 - ElementAssembly, [400](#), [401](#)
 - ElementNodeCoordinates, [400](#)
 - ElementNodeVector, [398](#)
 - ElementNodeVectorSingleDOF, [399](#)
 - ElementSideVector, [399](#)
 - ElementVector, [399](#)
 - getEigenValue, [405](#)
 - getEigenVector, [405](#)
 - getMesh, [404](#)
 - LocalNodeVector, [398](#)
 - setMaterialProperty, [404](#)
 - setSolver, [404](#)
 - SideAssembly, [403](#)
 - SideNodeCoordinates, [400](#)
 - SideVector, [400](#)
 - solveEigenProblem, [405](#)
 - updateBC, [397](#)
- OFELI::Equa_Fluid
 - AxbAssembly, [416](#), [417](#)
 - DGELEMENTAssembly, [414](#), [415](#)
 - DiagBC, [409](#)
 - ElementAssembly, [413](#), [414](#)
 - ElementNodeCoordinates, [413](#)
 - ElementNodeVector, [410](#)
 - ElementNodeVectorSingleDOF, [410](#)
 - ElementSideVector, [412](#)
 - ElementVector, [412](#)
 - Equa_Fluid, [409](#)
 - getEigenValue, [419](#)
 - getEigenVector, [419](#)
 - getMesh, [417](#)
 - LocalNodeVector, [410](#)
 - setMaterialProperty, [417](#)
 - setSolver, [418](#)
 - SideAssembly, [415](#), [416](#)
 - SideNodeCoordinates, [413](#)
 - SideVector, [412](#)
 - solveEigenProblem, [418](#)
 - updateBC, [409](#)
- OFELI::Equa_Laplace
 - AxbAssembly, [429](#), [430](#)
 - DGELEMENTAssembly, [427](#), [428](#)
 - DiagBC, [422](#)
 - ElementAssembly, [426](#), [427](#)
 - ElementNodeCoordinates, [426](#)
 - ElementNodeVector, [423](#)
 - ElementNodeVectorSingleDOF, [423](#)
 - ElementSideVector, [425](#)
 - ElementVector, [425](#)
 - Equa_Laplace, [422](#)

- getEigenValue, [432](#)
- getEigenvector, [432](#)
- getMesh, [430](#)
- LocalNodeVector, [423](#)
- setMaterialProperty, [430](#)
- setSolver, [431](#)
- SideAssembly, [428](#), [429](#)
- SideNodeCoordinates, [426](#)
- SideVector, [425](#)
- solveEigenProblem, [431](#)
- updateBC, [422](#)
- OFELI::Equa_Solid
 - AxbAssembly, [445](#)
 - buildEigen, [437](#)
 - DGElementAssembly, [443](#)
 - DiagBC, [438](#)
 - ElementAssembly, [440](#), [442](#)
 - ElementNodeCoordinates, [440](#)
 - ElementNodeVector, [438](#), [439](#)
 - ElementNodeVectorSingleDOF, [439](#)
 - ElementSideVector, [439](#)
 - ElementVector, [439](#)
 - Equa_Solid, [436](#)
 - getEigenValue, [446](#)
 - getEigenvector, [446](#)
 - getMesh, [445](#)
 - LMassToLHS, [436](#)
 - LMassToRHS, [437](#)
 - LocalNodeVector, [438](#)
 - MassToLHS, [437](#)
 - MassToRHS, [437](#)
 - setMaterialProperty, [445](#)
 - setSolver, [445](#)
 - SideAssembly, [444](#)
 - SideNodeCoordinates, [440](#)
 - SideVector, [440](#)
 - solveEigenProblem, [446](#)
 - updateBC, [437](#), [438](#)
- OFELI::Equa_Therm
 - AxbAssembly, [459](#)
 - build, [452](#)
 - buildEigen, [452](#)
 - CapacityToLHS, [451](#)
 - CapacityToRHS, [451](#)
 - DGElementAssembly, [457](#), [458](#)
 - DiagBC, [453](#)
 - ElementAssembly, [456](#), [457](#)
 - ElementNodeCoordinates, [455](#)
 - ElementNodeVector, [453](#), [454](#)
 - ElementNodeVectorSingleDOF, [454](#)
 - ElementSideVector, [454](#)
 - ElementVector, [454](#)
 - Equa_Therm, [451](#)
 - getEigenValue, [461](#)
 - getEigenvector, [461](#)
 - getMesh, [459](#)
 - LCapacityToLHS, [451](#)
 - LCapacityToRHS, [451](#)
 - LocalNodeVector, [453](#)
 - run, [452](#)
 - runTransient, [452](#)
 - setMaterialProperty, [459](#)
 - setSolver, [460](#)
 - setStab, [451](#)
 - SideAssembly, [458](#), [459](#)
 - SideNodeCoordinates, [455](#)
 - SideVector, [455](#)
 - solveEigenProblem, [461](#)
 - updateBC, [452](#), [453](#)
- OFELI::Equation
 - AxbAssembly, [472](#)
 - DGElementAssembly, [469](#), [470](#)
 - DiagBC, [466](#)
 - ElementAssembly, [468](#), [469](#), [471](#)
 - ElementNodeCoordinates, [468](#)
 - ElementNodeVector, [466](#), [467](#)
 - ElementNodeVectorSingleDOF, [466](#)
 - ElementSideVector, [467](#)
 - ElementVector, [467](#)
 - Equation, [464](#), [465](#)
 - getEigenValue, [473](#)
 - getEigenvector, [473](#)
 - getMesh, [472](#)
 - LocalNodeVector, [466](#)
 - setMaterialProperty, [472](#)
 - setSolver, [472](#)
 - SideAssembly, [470](#), [471](#)
 - SideNodeCoordinates, [468](#)
 - SideVector, [468](#)
 - solveEigenProblem, [473](#)
 - updateBC, [465](#)
- OFELI::FEShape
 - DSh, [478](#)
 - FEShape, [477](#), [478](#)
 - getDet, [478](#)
 - getLocalPoint, [478](#)
 - Sh, [478](#)
- OFELI::FMM2D
 - check_error, [481](#)
 - Evaluate, [481](#)
 - ExtendSpeed, [481](#)
 - FMM2D, [480](#)
 - InitHeap, [481](#)
- OFELI::FMM3D
 - check_error, [483](#)
 - Evaluate, [482](#)
 - ExtendSpeed, [483](#)
 - FMM3D, [482](#)

- InitHeap, 482
- OFELI::FMMSolver
 - check_error, 484
 - ExtendSpeed, 484
 - FMMSolver, 483
- OFELI::FastMarching2D
 - execute, 475
 - FastMarching2D, 475
- OFELI::Figure
 - dLine, 480
 - getSignedDistance, 479
- OFELI::Funct
 - Funct, 485
 - operator=, 486
- OFELI::Gauss
 - setTriangle, 486
- OFELI::Grid
 - Deactivate, 493
 - getCode, 493
 - getNy, 492
 - getNz, 492
 - Grid, 489, 490
 - isActive, 495
 - setCode, 492
 - setDomain, 491, 492
 - setN, 492
 - setXMax, 491
 - setXMin, 491
- OFELI::HelmholtzBT3
 - AxbAssembly, 505, 506
 - DGELEMENTAssembly, 503, 504
 - DiagBC, 499
 - ElementAssembly, 502, 503
 - ElementNodeCoordinates, 502
 - ElementNodeVector, 500
 - ElementNodeVectorSingleDOF, 500
 - ElementSideVector, 501
 - ElementVector, 501
 - getEigenValue, 508
 - getEigenVector, 508
 - getMesh, 506
 - LocalNodeVector, 500
 - setMaterialProperty, 506
 - setSolver, 507
 - SideAssembly, 504, 505
 - SideNodeCoordinates, 502
 - SideVector, 501
 - solveEigenProblem, 507
 - updateBC, 499
- OFELI::Hexa8
 - DSh, 509
 - getDet, 510
 - getLocalPoint, 510
 - setLocal, 509

- Sh, 509
- OFELI::ICPG1D
 - Forward, 514
 - getInternalEnergy, 514
 - getMach, 515
 - getMomentum, 514
 - getSoundSpeed, 515
 - getTotalEnergy, 515
 - ICPG1D, 514
 - Limiter, 513
 - Method, 513
 - setBC, 515, 516
 - setCFL, 517
 - setInOutflowBC, 516
 - setInitialCondition, 515
 - setLimiter, 517
 - setMethod, 517
 - setReconstruction, 517
 - setReferenceLength, 517
 - setTimeStep, 517
 - setVerbose, 517
 - SolverType, 513
- OFELI::ICPG2DT
 - Forward, 522
 - getP, 523
 - getR, 523
 - getV, 523
 - ICPG2DT, 521
 - Initialize, 524
 - Limiter, 521
 - Method, 521
 - setBC, 522, 523
 - setCFL, 524
 - setLimiter, 525
 - setMethod, 524
 - setReconstruction, 522, 524
 - setReferenceLength, 524
 - setSolver, 522
 - setTimeStep, 524
 - setVerbose, 524
 - SolverType, 521
- OFELI::ICPG3DT
 - ICPG3DT, 529, 530
 - Limiter, 528
 - Method, 528
 - setLimiter, 530
 - setMethod, 530
 - setReconstruction, 530
 - setVerbose, 530
 - SolverType, 528
- OFELI::IOField
 - get, 534
 - IOField, 532, 533
 - open, 534

- put, [534](#)
- saveGMSH, [534](#)
- setMeshFile, [533](#)
- OFELI::IPF
 - get, [540](#), [541](#)
 - getArraySize, [540](#)
 - getAuxFile, [542](#)
 - getBC, [538](#)
 - getBCFile, [541](#)
 - getBF, [538](#)
 - getBFFile, [542](#)
 - getComplex, [540](#)
 - getComplexPar, [539](#)
 - getData, [538](#)
 - getDouble, [540](#)
 - getDoublePar, [539](#)
 - getInit, [538](#)
 - getInitFile, [541](#)
 - getIntPar, [539](#)
 - getInteger, [539](#)
 - getMaxTime, [538](#)
 - getMeshFile, [541](#)
 - getNblter, [538](#)
 - getNbSteps, [538](#)
 - getOutput, [537](#)
 - getPlot, [538](#)
 - getPlotFile, [542](#)
 - getPointDoublePar, [539](#)
 - getPrescriptionFile, [542](#)
 - getProject, [541](#)
 - getRestartFile, [541](#)
 - getSF, [538](#)
 - getSFFile, [542](#)
 - getSave, [537](#)
 - getSaveFile, [542](#)
 - getString, [539](#)
 - getStringPar, [539](#)
 - getTimeStep, [538](#)
 - getTolerance, [538](#)
 - IPF, [537](#)
- OFELI::Iter
 - check, [543](#)
 - Iter, [543](#)
- OFELI::LCL1D
 - Forward, [616](#)
 - Limiter, [614](#)
 - Method, [614](#)
 - runOneTimeStep, [615](#)
 - setBC, [615](#), [616](#)
 - setCFL, [616](#)
 - setInitialCondition, [615](#)
 - setLimiter, [617](#)
 - setMethod, [617](#)
 - setReconstruction, [617](#)

- setTimeStep, [616](#)
- setVelocity, [616](#)
- setVerbose, [617](#)
- SolverType, [615](#)
- OFELI::LCL2DT
 - Forward, [622](#)
 - Initialize, [622](#)
 - LCL2DT, [620](#)
 - Limiter, [620](#)
 - Method, [620](#)
 - runOneTimeStep, [621](#)
 - setBC, [621](#)
 - setCFL, [622](#)
 - setInitialCondition, [621](#)
 - setLimiter, [623](#)
 - setMethod, [623](#)
 - setReconstruction, [622](#)
 - setReferenceLength, [622](#)
 - setTimeStep, [622](#)
 - setVelocity, [621](#), [622](#)
 - setVerbose, [623](#)
 - SolverType, [620](#)
- OFELI::LCL3DT
 - Forward, [628](#)
 - LCL3DT, [626](#)
 - Limiter, [626](#)
 - Method, [626](#)
 - setBC, [627](#)
 - setCFL, [628](#)
 - setInitialCondition, [627](#)
 - setLimiter, [628](#)
 - setMethod, [628](#)
 - setReconstruction, [628](#)
 - setTimeStep, [628](#)
 - setVelocity, [627](#)
 - setVerbose, [628](#)
 - SolverType, [626](#)
- OFELI::Laplace1DL2
 - AxbAssembly, [555](#), [556](#)
 - BodyRHS, [548](#)
 - BoundaryRHS, [548](#)
 - DGElementAssembly, [553](#), [554](#)
 - DiagBC, [549](#)
 - ElementAssembly, [552](#), [553](#)
 - ElementNodeCoordinates, [552](#)
 - ElementNodeVector, [550](#)
 - ElementNodeVectorSingleDOF, [550](#)
 - ElementSideVector, [550](#)
 - ElementVector, [551](#)
 - getEigenValue, [558](#)
 - getEigenVector, [558](#)
 - getMesh, [556](#)
 - Laplace1DL2, [547](#)
 - LocalNodeVector, [549](#)

- Matrix, 548
- run, 548
- setBoundaryCondition, 548
- setMaterialProperty, 556
- setSolver, 557
- setTraction, 548
- SideAssembly, 554, 555
- SideNodeCoordinates, 552
- SideVector, 551
- solveEigenProblem, 557
- updateBC, 549
- OFELI::Laplace1DL3
 - AxbAssembly, 568, 569
 - BodyRHS, 561
 - BoundaryRHS, 561
 - DGElementAssembly, 566, 567
 - DiagBC, 562
 - ElementAssembly, 565, 566
 - ElementNodeCoordinates, 565
 - ElementNodeVector, 563
 - ElementNodeVectorSingleDOF, 563
 - ElementSideVector, 564
 - ElementVector, 564
 - getEigenValue, 571
 - getEigenVector, 571
 - getMesh, 569
 - Laplace1DL3, 561
 - LocalNodeVector, 563
 - Matrix, 561
 - run, 562
 - setMaterialProperty, 569
 - setSolver, 570
 - setTraction, 562
 - SideAssembly, 567, 568
 - SideNodeCoordinates, 565
 - SideVector, 564
 - solveEigenProblem, 570
 - updateBC, 562
- OFELI::Laplace2DFVT
 - AxbAssembly, 581, 582
 - checkDelaunay, 574
 - DGElementAssembly, 579, 580
 - DiagBC, 575
 - ElementAssembly, 578, 579
 - ElementNodeCoordinates, 578
 - ElementNodeVector, 576
 - ElementNodeVectorSingleDOF, 576
 - ElementSideVector, 577
 - ElementVector, 577
 - getEigenValue, 584
 - getEigenVector, 584
 - getMesh, 582
 - Laplace2DFVT, 574
 - LocalNodeVector, 576

- setMaterialProperty, 582
- setSolver, 583
- SideAssembly, 580, 581
- SideNodeCoordinates, 578
- SideVector, 577
- solveEigenProblem, 583
- updateBC, 575
- OFELI::Laplace2DMHRT0
 - AxbAssembly, 595
 - build, 588
 - DGElementAssembly, 593
 - DiagBC, 589
 - ElementAssembly, 591, 592
 - ElementNodeCoordinates, 591
 - ElementNodeVector, 589
 - ElementNodeVectorSingleDOF, 590
 - ElementSideVector, 590
 - ElementVector, 590
 - getEigenValue, 596
 - getEigenVector, 596
 - getMesh, 595
 - Laplace2DMHRT0, 587
 - LocalNodeVector, 589
 - Post, 588
 - setDiffusivity, 587
 - setMaterialProperty, 595
 - setSolver, 595
 - SideAssembly, 594
 - SideNodeCoordinates, 591
 - SideVector, 591
 - solve, 588
 - solveEigenProblem, 596
 - updateBC, 588
- OFELI::Laplace2DT3
 - Axb, 602
 - AxbAssembly, 609, 610
 - BodyRHS, 601
 - BoundaryRHS, 601
 - build, 602
 - buildEigen, 602
 - DGElementAssembly, 607, 608
 - DiagBC, 603
 - ElementAssembly, 606, 607
 - ElementNodeCoordinates, 606
 - ElementNodeVector, 604
 - ElementNodeVectorSingleDOF, 604
 - ElementSideVector, 604
 - ElementVector, 605
 - getEigenValue, 612
 - getEigenVector, 612
 - getMesh, 610
 - LHS, 601
 - Laplace2DT3, 600, 601
 - LocalNodeVector, 603

- Post, 602
- setDiffusivity, 601
- setMaterialProperty, 610
- setSolver, 611
- setSource, 601
- SideAssembly, 608, 609
- SideNodeCoordinates, 606
- SideVector, 605
- solve, 602
- solveEigenProblem, 611
- updateBC, 603
- OFELI::Line2
 - DSh, 631
 - getDet, 631
 - getInterpolate, 631
 - getLocalPoint, 631
 - getRefCoord, 631
 - Line2, 630
 - Sh, 630, 631
- OFELI::Line2H
 - check, 633
 - DSh, 633
 - getLocalPoint, 633
 - Sh, 633
- OFELI::Line3
 - getDet, 634
 - getLocalPoint, 634
 - Sh, 634
- OFELI::LocalMatrix
 - Factor, 640
 - FactorAndSolve, 640
 - getInnerProduct, 641
 - Invert, 641
 - LocalMatrix, 636, 637
 - Localize, 637, 638
 - Mult, 640
 - MultAdd, 638
 - MultAddScal, 640
 - operator*, 638
 - operator*==, 638
 - operator(), 637
 - operator+=, 638
 - operator-=, 638
 - operator/==, 638
 - operator=, 638
 - Solve, 640
 - Symmetrize, 640
- OFELI::LocalVect
 - getLocal, 644
 - LocalVect, 643
 - Localize, 644
 - operator*==, 645
 - operator(), 644
 - operator+=, 645

- operator,, 645
- operator-=, 645
- operator/==, 645
- operator=, 645
- OFELI::Material
 - check, 648
 - getName, 648
 - Material, 648
 - set, 648
- OFELI::Matrix
 - Assembly, 651, 652
 - Axpy, 651
 - FactorAndSolve, 655, 656
 - getDiag, 651
 - getPrintView, 658
 - isFactorized, 656
 - Matrix, 651
 - operator*==, 657
 - operator(), 656, 657
 - operator+=, 657, 658
 - operator-=, 657, 658
 - operator=, 657
 - Prescribe, 652–654
 - PrescribeSide, 655
 - set, 656
 - setDiagonal, 651
 - setPrintView, 658
 - Solve, 655
- OFELI::Mesh
 - Add, 667, 670
 - AddMidNodes, 674
 - Bput, 675
 - createBoundarySideList, 669
 - createInternalSideList, 669
 - Delete, 671, 672
 - DeleteElement, 671
 - DeleteNode, 671
 - DeleteSide, 671
 - EdgesAreDOF, 675
 - ElementsAreDOF, 675
 - get, 667, 668
 - getActiveElement, 677
 - getAllEdges, 670
 - getAllSides, 669
 - getBoundaryNodes, 669
 - getBoundarySides, 669
 - getEdgeLabel, 676
 - getElementLabel, 676
 - getElementNeighborElements, 670
 - getList, 676
 - getNbBoundarySides, 674
 - getNbInternalSides, 674
 - getNodeLabel, 676
 - getNodeNeighborElements, 670

- getSideLabel, 676
- Mesh, 664–666
- NodesAreDOF, 675
- NumberEquations, 669
- operator*=, 667
- put, 675
- Refine, 677
- RenumberEdge, 672
- RenumberElement, 672
- RenumberNode, 672
- RenumberSide, 672
- Reorder, 670
- Rescale, 673
- save, 675
- set, 674
- setDOFSupport, 668
- setDim, 666
- setEdgeView, 673
- setElementView, 673
- setList, 673
- setMaterial, 670
- setNbDOFPerNode, 668
- setNodeView, 672
- setPointInDomain, 668
- setSideView, 673
- setVerbose, 667
- SidesAreDOF, 675
- OFELI::MeshAdapt
 - getSolution, 680
 - getSolutionBB, 680
 - getSolutionbb, 680
 - MeshAdapt, 679
 - saveMbb, 681
 - setMaxNbVertices, 680
 - setNoKeep, 680
 - setNoScaling, 680
 - setRatio, 680
 - setRelaxation, 680
 - setTheta, 681
- OFELI::Muscl
 - Limiter, 683
 - Method, 683
 - setCFL, 684
 - setLimiter, 684
 - setMethod, 684
 - setReconstruction, 684
 - setReferenceLength, 684
 - setTimeStep, 683
 - setVerbose, 684
 - SolverType, 683
- OFELI::Muscl1D
 - Limiter, 687
 - Method, 687
 - setCFL, 688

- setLimiter, 688
 - setMethod, 688
 - setReconstruction, 688
 - setReferenceLength, 688
 - setTimeStep, 687
 - setVerbose, 688
 - SolverType, 687
- OFELI::Muscl2DT
 - Initialize, 691
 - Limiter, 690
 - Method, 690
 - setCFL, 692
 - setLimiter, 692
 - setMethod, 692
 - setReconstruction, 691
 - setReferenceLength, 692
 - setTimeStep, 691
 - setVerbose, 692
 - SolverType, 691
- OFELI::Muscl3DT
 - Limiter, 695
 - Method, 695
 - setCFL, 696
 - setLimiter, 696
 - setMethod, 696
 - setReconstruction, 695
 - setReferenceLength, 696
 - setTimeStep, 696
 - setVerbose, 696
 - SolverType, 695
- OFELI::NSP2DQ41
 - AxbAssembly, 714
 - BodyRHS, 707
 - BoundaryRHS, 707
 - DGELEMENTAssembly, 712, 713
 - DiagBC, 708
 - ElementAssembly, 711, 712
 - ElementNodeCoordinates, 710
 - ElementNodeVector, 708, 709
 - ElementNodeVectorSingleDOF, 709
 - ElementSideVector, 709
 - ElementVector, 709
 - getEigenValue, 716
 - getEigenVector, 716
 - getMesh, 714
 - LHS1.Convection, 707
 - LHS2.Convection, 707
 - LocalNodeVector, 708
 - NSP2DQ41, 706, 707
 - Periodic, 707
 - setMaterialProperty, 714
 - setSolver, 715
 - SideAssembly, 713, 714
 - SideNodeCoordinates, 710

- SideVector, 710
- solveEigenProblem, 716
- updateBC, 707, 708
- OFELI::Node
 - DOF, 699
 - getCode, 700
 - getCoord, 700
 - getNbNeigEl, 700
 - getNeigEl, 700
 - getXYZ, 700
 - isOnBoundary, 700
 - Node, 698
 - setCode, 699
 - setCoord, 699
 - setDOF, 700
 - setLevel, 700
 - setOnBoundary, 700
- OFELI::NodeList
 - selectCode, 701
 - selectCoordinate, 702
 - unselectCode, 702
- OFELI::Partition
 - getSecondSideLabel, 717
 - Partition, 717
- OFELI::Penta6
 - DSh, 719
 - getDet, 719
 - getLocalPoint, 719, 720
 - Penta6, 719
 - setLocal, 719
 - Sh, 719
- OFELI::PhaseChange
 - E2T, 720
 - EnthalpyToTemperature, 721
- OFELI::Point
 - isCloseTo, 724
 - Normalize, 723
 - operator*, 723
 - operator(), 722
 - operator+=, 723
 - operator,, 724
 - operator=, 723
 - operator/ =, 723
 - operator=, 723
 - operator==, 723
 - Point, 722
- OFELI::Point2D
 - operator*, 726
 - operator(), 725
 - operator+=, 726
 - operator=, 726
 - operator/ =, 726
 - operator=, 726
 - operator==, 726

- Point2D, 725
- OFELI::Polygon
 - dLine, 728
 - getSignedDistance, 728
 - operator+=, 728
 - Polygon, 727
 - setVertices, 728
- OFELI::Prec
 - Prec, 730
 - setMatrix, 731
 - setType, 731
 - Solve, 731
 - TransSolve, 731, 732
- OFELI::Prescription
 - get, 733
 - Prescription, 733
- OFELI::Quad4
 - DSh, 735
 - getDet, 736
 - getLocalPoint, 736
 - Grad, 735
 - Quad4, 735
 - setLocal, 735
 - Sh, 736
- OFELI::Reconstruction
 - DP1toP1, 737
 - P0toP1, 737
- OFELI::Rectangle
 - dLine, 739
 - getSignedDistance, 738, 739
 - operator+=, 739
 - Rectangle, 738
 - setBoundingBox, 738
- OFELI::Side
 - Add, 743
 - Contains, 745
 - DOF, 742
 - getCode, 743
 - getMeasure, 744
 - getNeighborElement, 744
 - getNormal, 744
 - getOtherNeighborElement, 744
 - getUnitNormal, 744
 - isOnBoundary, 744
 - operator(), 744
 - set, 743
 - setChild, 745
 - setCode, 743
 - setDOF, 742
 - Side, 742
 - SideType, 742
- OFELI::SideList
 - selectCode, 746
 - unselectCode, 746

OFELI::SkMatrix
 add, 752
 Assembly, 755, 756
 Axy, 751
 DiagPrescribe, 753
 Factor, 754
 FactorAndSolve, 759
 get, 755
 getColHeight, 753
 getDiag, 755
 getPrintView, 760
 isFactorized, 759
 Mult, 752
 MultAdd, 752
 operator*=, 754
 operator(), 753, 759
 operator+=, 754, 760
 operator=, 760
 operator=, 754
 Prescribe, 756–758
 PrescribeSide, 758
 set, 751
 setDOF, 751
 setDiagonal, 755
 setMesh, 750
 setPrintView, 760
 setSkyline, 751
 SkMatrix, 750
 Solve, 754, 755
 TMult, 752
 TMultAdd, 752
 OFELI::SkSMatrix
 add, 767
 Assembly, 769, 770
 Axy, 766
 Factor, 768
 FactorAndSolve, 773
 get, 769
 getColHeight, 767
 getDiag, 769
 getPrintView, 774
 isFactorized, 773
 Mult, 767
 MultAdd, 766, 767
 operator*=, 768
 operator(), 767, 768, 773
 operator+=, 768, 774
 operator=, 774
 operator=, 768
 Prescribe, 770–772
 PrescribeSide, 772
 set, 766
 setDiagonal, 769
 setMesh, 766

 setPrintView, 774
 setSkyline, 766
 SkSMatrix, 764, 765
 Solve, 768, 769
 TMult, 767
 OFELI::SpMatrix
 add, 787
 Assembly, 789, 790
 Axy, 786
 DiagPrescribe, 784
 FactorAndSolve, 793
 get, 789
 getColInd, 787
 getDiag, 789
 getPrintView, 794
 isFactorized, 793
 Mult, 786
 MultAdd, 786
 operator*=, 794
 operator(), 785, 793
 operator+=, 793
 operator=, 794
 operator=, 787
 Prescribe, 790–792
 PrescribeSide, 792
 set, 787
 setDiagonal, 789
 setGraph, 785
 setMesh, 784
 setPrintView, 794
 setSize, 785
 setSolver, 788
 Solve, 787, 788
 SpMatrix, 782–784
 TMult, 786
 OFELI::Sphere
 dLine, 777
 getSignedDistance, 777
 operator+=, 777
 Sphere, 776
 OFELI::SteklovPoincare2DBE
 setMesh, 796
 Solve, 796
 SteklovPoincare2DBE, 795
 OFELI::TINS2DT3B
 AxbAssembly, 813
 DGELEMENTAssembly, 811, 812
 DiagBC, 807
 ElementAssembly, 810, 811
 ElementNodeCoordinates, 809
 ElementNodeVector, 807, 808
 ElementNodeVectorSingleDOF, 808
 ElementSideVector, 808
 ElementVector, 808

- getEigenValue, 815
- getEigenvector, 815
- getMesh, 813
- LocalNodeVector, 807
- setInput, 805
- setMaterialProperty, 813
- setSolver, 814
- SideAssembly, 812, 813
- SideNodeCoordinates, 809
- SideVector, 809
- solveEigenProblem, 815
- TINS2DT3B, 805
- updateBC, 805, 807
- OFELI::Tabulation
 - getDerivative, 797
 - getValue, 797, 798
 - setFile, 797
- OFELI::Tetra4
 - CurlEdgeSh, 800
 - DSh, 799
 - EdgeSh, 799
 - getDet, 800
 - getLocalPoint, 800
 - Sh, 799
- OFELI::Timer
 - getTime, 801
 - Start, 801
 - Started, 801
 - Stop, 801
- OFELI::TrMatrix
 - Assembly, 829, 830
 - Axpy, 828
 - FactorAndSolve, 833
 - getDiag, 829
 - getPrintView, 835
 - isFactorized, 834
 - operator*=, 829
 - operator(), 828, 829, 834
 - operator+=, 834
 - operator-=, 834, 835
 - operator=, 829
 - Prescribe, 830–832
 - PrescribeSide, 833
 - setDiagonal, 829
 - setPrintView, 835
 - setSize, 828
 - Solve, 829, 833
 - TrMatrix, 828
- OFELI::Triang3
 - check, 817
 - getDet, 817
 - getLocalPoint, 817
 - Grad, 817
 - Sh, 817
 - Triang3, 817
- OFELI::Triang6S
 - DSh, 819, 820
 - getDet, 820
 - getLocalPoint, 820
 - Grad, 819
 - Sh, 819, 820
 - Triang6S, 819
- OFELI::UserData
 - BodyForce, 836
 - BoundaryCondition, 836
 - InitialData, 836
 - setBodyForce, 836
 - SurfaceForce, 836
- OFELI::Vect
 - add, 853, 854
 - Assembly, 850, 851
 - Axpy, 853
 - getAverage, 852
 - getCurl, 851
 - getDOFType, 846
 - getDivergence, 852
 - getGradient, 851
 - getPrintView, 856
 - getSCurl, 852
 - getWNorm2, 846
 - insertBC, 849, 850
 - MultAdd, 852
 - operator*=, 856
 - operator(), 854, 855
 - operator+=, 855
 - operator,, 856
 - operator-=, 856
 - operator/=, 856
 - operator=, 855
 - removeBC, 848
 - resize, 846
 - save, 852
 - select, 844
 - set, 844, 845, 853
 - setDOFType, 846
 - setMesh, 845
 - setNodeBC, 846, 848
 - setPrintView, 856
 - setSize, 845
 - transferBC, 849
 - Vect, 841–844
 - WithMesh, 846
- OFELI::triangle
 - DSh, 822, 824
 - getDet, 822, 824
 - getLocalPoint, 822, 824
 - Sh, 822, 824
 - triangle, 822, 824

- OFELI.E
 - Utilities, 70
- OFELI.EPSMCH
 - Utilities, 70
- OFELI.GAUSS2
 - Utilities, 70
- OFELI.IMAG
 - Utilities, 70
- OFELI.ONEOVERPI
 - Utilities, 70
- OFELI.PI
 - Utilities, 70
- OFELI.SIXTH
 - Utilities, 70
- OFELI.SQRT2
 - Utilities, 70
- OFELI.SQRT3
 - Utilities, 70
- OFELI.THIRD
 - Utilities, 70
- OFELI.TOLERANCE
 - Utilities, 70
- OFELI.TWELVETH
 - Utilities, 70
- open
 - OFELI::IOField, 534
- operator*
 - OFELI::LocalMatrix, 638
 - Utilities, 72–74
 - Vector and Matrix, 86, 87
- operator*=
 - OFELI::BMatrix, 157
 - OFELI::DMatrix, 272
 - OFELI::Domain, 281
 - OFELI::DSMatrix, 294
 - OFELI::LocalMatrix, 638
 - OFELI::LocalVect, 645
 - OFELI::Matrix, 657
 - OFELI::Mesh, 667
 - OFELI::Point, 723
 - OFELI::Point2D, 726
 - OFELI::SkMatrix, 754
 - OFELI::SkSMatrix, 768
 - OFELI::SpMatrix, 794
 - OFELI::TrMatrix, 829
 - OFELI::Vect, 856
- operator()
 - OFELI::BMatrix, 156, 162
 - OFELI::DMatrix, 271, 277
 - OFELI::DSMatrix, 286, 293, 294
 - OFELI::Edge, 311
 - OFELI::Element, 389, 390
 - OFELI::LocalMatrix, 637
 - OFELI::LocalVect, 644
 - OFELI::Matrix, 656, 657
 - OFELI::Point, 722
 - OFELI::Point2D, 725
 - OFELI::Side, 744
 - OFELI::SkMatrix, 753, 759
 - OFELI::SkSMatrix, 767, 768, 773
 - OFELI::SpMatrix, 785, 793
 - OFELI::TrMatrix, 828, 829, 834
 - OFELI::Vect, 854, 855
- operator+
 - Utilities, 72, 73
 - Vector and Matrix, 86
- operator+=
 - OFELI::BMatrix, 157, 162
 - OFELI::Brick, 164
 - OFELI::Circle, 166
 - OFELI::DMatrix, 272, 278
 - OFELI::DSMatrix, 294
 - OFELI::Ellipse, 393
 - OFELI::LocalMatrix, 638
 - OFELI::LocalVect, 645
 - OFELI::Matrix, 657, 658
 - OFELI::Point, 723
 - OFELI::Point2D, 726
 - OFELI::Polygon, 728
 - OFELI::Rectangle, 739
 - OFELI::SkMatrix, 754, 760
 - OFELI::SkSMatrix, 768, 774
 - OFELI::Sphere, 777
 - OFELI::SpMatrix, 793
 - OFELI::TrMatrix, 834
 - OFELI::Vect, 855
- operator,
 - OFELI::LocalVect, 645
 - OFELI::Point, 724
 - OFELI::Vect, 856
- operator-
 - Finite Element Mesh, 23
 - Utilities, 72, 73
 - Vector and Matrix, 86, 87
- operator-=
 - OFELI::BMatrix, 162
 - OFELI::DMatrix, 272, 278
 - OFELI::DSMatrix, 294
 - OFELI::LocalMatrix, 638
 - OFELI::LocalVect, 645
 - OFELI::Matrix, 657, 658
 - OFELI::Point, 723
 - OFELI::Point2D, 726
 - OFELI::SkMatrix, 760
 - OFELI::SkSMatrix, 774
 - OFELI::SpMatrix, 794
 - OFELI::TrMatrix, 834, 835
 - OFELI::Vect, 856

- operator/
 - Utilities, [73](#), [74](#)
 - Vector and Matrix, [86](#), [87](#)
- operator/=
 - OFELI::LocalMatrix, [638](#)
 - OFELI::LocalVect, [645](#)
 - OFELI::Point, [723](#)
 - OFELI::Point2D, [726](#)
 - OFELI::Vect, [856](#)
- operator=
 - OFELI::BMatrix, [156](#)
 - OFELI::DMatrix, [272](#)
 - OFELI::Funct, [486](#)
 - OFELI::LocalMatrix, [638](#)
 - OFELI::LocalVect, [645](#)
 - OFELI::Matrix, [657](#)
 - OFELI::Point, [723](#)
 - OFELI::Point2D, [726](#)
 - OFELI::SkMatrix, [754](#)
 - OFELI::SkSMatrix, [768](#)
 - OFELI::SpMatrix, [787](#)
 - OFELI::TrMatrix, [829](#)
 - OFELI::Vect, [855](#)
- operator==
 - OFELI::Point, [723](#)
 - OFELI::Point2D, [726](#)
 - Utilities, [72](#), [73](#)
- operator&&
 - Finite Element Mesh, [23](#)
- OptimSA
 - Solver, [54](#)
- OptimTN
 - Solver, [57](#)
- P0toP1
 - OFELI::Reconstruction, [737](#)
- PERIODIC.A
 - OFELI, [111](#)
- PERIODIC.B
 - OFELI, [111](#)
- PLANAR.TRUSS
 - OFELI, [109](#)
- POINT_FORCE
 - OFELI, [108](#)
- PARSE
 - Utilities, [70](#)
- PDE
 - OFELI, [109](#)
- PDE_Terms
 - OFELI, [108](#)
- Partition, [716](#)
 - OFELI::Partition, [717](#)
- Penta6, [718](#)
 - OFELI::Penta6, [719](#)
- Periodic
 - OFELI::DC2DT3, [197](#)
 - OFELI::DC3DT4, [254](#)
 - OFELI::Elas2DT3, [343](#)
 - OFELI::NSP2DQ41, [707](#)
- PhaseChange, [720](#)
- Physical properties of media, [43](#)
- PlaneStrain
 - OFELI::Elas2DQ4, [321](#)
- PlaneStress
 - OFELI::Elas2DQ4, [321](#)
- Point
 - OFELI::Point, [722](#)
- Point< T_ >, [721](#)
- Point2D
 - OFELI::Point2D, [725](#)
- Point2D< T_ >, [724](#)
- Polygon, [727](#)
 - OFELI::Polygon, [727](#)
- Post
 - OFELI::Laplace2DMHRT0, [588](#)
 - OFELI::Laplace2DT3, [602](#)
- Prec
 - OFELI::Prec, [730](#)
- Prec< T_ >, [729](#)
- Precond< T_ >, [732](#)
- Preconditioner
 - OFELI, [110](#)
- Prescribe
 - OFELI::BMatrix, [158–160](#)
 - OFELI::DMatrix, [274–276](#)
 - OFELI::DSMatrix, [290–292](#)
 - OFELI::Matrix, [652–654](#)
 - OFELI::SkMatrix, [756–758](#)
 - OFELI::SkSMatrix, [770–772](#)
 - OFELI::SpMatrix, [790–792](#)
 - OFELI::TrMatrix, [830–832](#)
- PrescribeSide
 - OFELI::BMatrix, [161](#)
 - OFELI::DMatrix, [276](#)
 - OFELI::DSMatrix, [293](#)
 - OFELI::Matrix, [655](#)
 - OFELI::SkMatrix, [758](#)
 - OFELI::SkSMatrix, [772](#)
 - OFELI::SpMatrix, [792](#)
 - OFELI::TrMatrix, [833](#)
- Prescription, [732](#)
 - OFELI::Prescription, [733](#)
- Prev
 - OFELI, [111](#)
- put
 - OFELI::IOField, [534](#)
 - OFELI::Mesh, [675](#)

- QMR_SOLVER
 - OFELI, [110](#)
- QMR
 - Solver, [58](#)
- qksort
 - Utilities, [81](#)
- Quad4, [734](#)
 - OFELI::Quad4, [735](#)
- QuickSort
 - Utilities, [81](#)
- RK4
 - OFELI, [109](#)
- ROE_SOLVER
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- RUNGE_KUTTA
 - OFELI, [109](#)
- RUSANOV_SOLVER
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- RHS_Convection
 - OFELI::DC3DT4, [253](#)
- Reaction
 - OFELI::Elas2DT3, [343](#)
- Reconstruction, [736](#)
- Rectangle, [737](#)
 - OFELI::Rectangle, [738](#)
- Refine
 - OFELI::Mesh, [677](#)
- removeBC
 - OFELI::Vect, [848](#)
- RenumberEdge
 - OFELI::Mesh, [672](#)
- RenumberElement
 - OFELI::Mesh, [672](#)
- RenumberNode
 - OFELI::Mesh, [672](#)
- RenumberSide
 - OFELI::Mesh, [672](#)
- Reorder
 - OFELI::Mesh, [670](#)
- Replace
 - OFELI::Element, [388](#)
- Rescale
 - OFELI::Mesh, [673](#)
- resize
 - OFELI::Vect, [846](#)
- Richardson
 - Solver, [59](#)
- run
 - OFELI::BiotSavart, [152](#)
 - OFELI::DC1DL2, [177](#)
 - OFELI::DC2DT3, [198](#)
 - OFELI::DC2DT6, [216](#)
 - OFELI::DC3DAT3, [234](#)
 - OFELI::DC3DT4, [255](#)
 - OFELI::Equa_Therm, [452](#)
 - OFELI::Laplace1DL2, [548](#)
 - OFELI::Laplace1DL3, [562](#)
- runOneTimeStep
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [621](#)
- runSteadyState
 - OFELI::DC2DT3, [198](#)
- runSubSpace
 - OFELI::Eigen, [315](#)
- runTransient
 - OFELI::DC1DL2, [177](#)
 - OFELI::DC2DT3, [198](#)
 - OFELI::DC2DT6, [216](#)
 - OFELI::DC3DAT3, [234](#)
 - OFELI::DC3DT4, [254](#)
 - OFELI::Equa_Therm, [452](#)
- SKYLINE
 - OFELI, [110](#)
- SLIP
 - OFELI, [111](#)
- SOLUTION
 - OFELI, [108](#)
- SOURCE
 - OFELI, [108](#)
- SPARSE
 - OFELI, [110](#)
- SPATIAL_BEAM
 - OFELI, [109](#)
- SSOR_PREC
 - OFELI, [111](#)
- STATIONARY
 - OFELI, [109](#)
- STEADY_STATE

OFELI, [110](#)
 STIFFNESS
 OFELI, [108](#)
 SUPERBEE_LIMITER
 OFELI::ICPG1D, [513](#)
 OFELI::ICPG2DT, [521](#)
 OFELI::ICPG3DT, [528](#)
 OFELI::LCL1D, [615](#)
 OFELI::LCL2DT, [620](#)
 OFELI::LCL3DT, [626](#)
 OFELI::Muscl, [683](#)
 OFELI::Muscl1D, [687](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 SYMMETRIC
 OFELI, [110](#)
 SSOR
 Solver, [61](#)
 save
 OFELI::Mesh, [675](#)
 OFELI::Vect, [852](#)
 saveBamg
 Utilities, [80](#)
 saveField
 Utilities, [71](#)
 saveGMSH
 OFELI::IOField, [534](#)
 saveGmsh
 Utilities, [72, 79](#)
 saveGnuplot
 Utilities, [71, 79](#)
 saveMatlab
 Utilities, [79](#)
 saveMbb
 OFELI::MeshAdapt, [681](#)
 saveMesh
 Utilities, [78](#)
 saveTecplot
 Utilities, [71, 79](#)
 saveVTK
 Utilities, [72, 80](#)
 Scale
 Utilities, [82](#)
 Schur
 Solver, [60](#)
 select
 OFELI::Vect, [844](#)
 selectCode
 OFELI::EdgeList, [312](#)
 OFELI::NodeList, [701](#)
 OFELI::SideList, [746](#)
 selectCoordinate
 OFELI::NodeList, [702](#)
 selectLevel

OFELI::ElementList, [392](#)
 set
 OFELI::DMatrix, [269](#)
 OFELI::DSMatrix, [286](#)
 OFELI::Element, [389](#)
 OFELI::Material, [648](#)
 OFELI::Matrix, [656](#)
 OFELI::Mesh, [674](#)
 OFELI::Side, [743](#)
 OFELI::SkMatrix, [751](#)
 OFELI::SkSMatrix, [766](#)
 OFELI::SpMatrix, [787](#)
 OFELI::Vect, [844, 845, 853](#)
 setBC
 OFELI::ICPG1D, [515, 516](#)
 OFELI::ICPG2DT, [522, 523](#)
 OFELI::LCL1D, [615, 616](#)
 OFELI::LCL2DT, [621](#)
 OFELI::LCL3DT, [627](#)
 setBodyForce
 OFELI::UserData, [836](#)
 setBoundary
 OFELI::BiotSavart, [150](#)
 setBoundaryCondition
 OFELI::Laplace1DL2, [548](#)
 setBoundaryNodeCodes
 Finite Element Mesh, [26](#)
 setBoundarySideCodes
 Finite Element Mesh, [27](#)
 setBoundingBox
 OFELI::Brick, [164](#)
 OFELI::Rectangle, [738](#)
 setCFL
 OFELI::ICPG1D, [517](#)
 OFELI::ICPG2DT, [524](#)
 OFELI::LCL1D, [616](#)
 OFELI::LCL2DT, [622](#)
 OFELI::LCL3DT, [628](#)
 OFELI::Muscl, [684](#)
 OFELI::Muscl1D, [688](#)
 OFELI::Muscl2DT, [692](#)
 OFELI::Muscl3DT, [696](#)
 setChild
 OFELI::Element, [391](#)
 OFELI::Side, [745](#)
 setCode
 OFELI::Edge, [311](#)
 OFELI::Element, [387, 389](#)
 OFELI::Grid, [492](#)
 OFELI::Node, [699](#)
 OFELI::Side, [743](#)
 setColumn
 OFELI::DMatrix, [270](#)
 setCoord

- OFELI::Node, 699
- setCurrentDensity
 - OFELI::BiotSavart, 150
- setDOF
 - OFELI::Edge, 311
 - OFELI::Element, 389
 - OFELI::Node, 700
 - OFELI::Side, 742
 - OFELI::SkMatrix, 751
- setDOFSupport
 - OFELI::Mesh, 668
- setDOFType
 - OFELI::Vect, 846
- setDiagonal
 - OFELI::BMatrix, 157
 - OFELI::DMatrix, 273
 - OFELI::DSMatrix, 288
 - OFELI::Matrix, 651
 - OFELI::SkMatrix, 755
 - OFELI::SkSMatrix, 769
 - OFELI::SpMatrix, 789
 - OFELI::TrMatrix, 829
- setDiffusivity
 - OFELI::Laplace2DMHRT0, 587
 - OFELI::Laplace2DT3, 601
- setDim
 - OFELI::Mesh, 666
- setDomain
 - OFELI::Grid, 491, 492
- setEdgeView
 - OFELI::Mesh, 673
- setElementCodes
 - Finite Element Mesh, 27
- setElementView
 - OFELI::Mesh, 673
- setFile
 - OFELI::Tabulation, 797
- setGraph
 - OFELI::SpMatrix, 785
- setInOutflowBC
 - OFELI::ICPG1D, 516
- setInitialCondition
 - OFELI::ICPG1D, 515
 - OFELI::LCL1D, 615
 - OFELI::LCL2DT, 621
 - OFELI::LCL3DT, 627
- setInput
 - OFELI::DC1DL2, 176
 - OFELI::DC2DT3, 197
 - OFELI::TINS2DT3B, 805
- setLabel
 - OFELI::Element, 387
- setLevel
 - OFELI::Node, 700

- setLimiter
 - OFELI::ICPG1D, 517
 - OFELI::ICPG2DT, 525
 - OFELI::ICPG3DT, 530
 - OFELI::LCL1D, 617
 - OFELI::LCL2DT, 623
 - OFELI::LCL3DT, 628
 - OFELI::Muscl, 684
 - OFELI::Muscl1D, 688
 - OFELI::Muscl2DT, 692
 - OFELI::Muscl3DT, 696
- setList
 - OFELI::Mesh, 673
- setLocal
 - OFELI::Hexa8, 509
 - OFELI::Penta6, 719
 - OFELI::Quad4, 735
- setMagneticInduction
 - OFELI::BiotSavart, 150
- setMaterial
 - OFELI::Mesh, 670
- setMaterialProperty
 - OFELI::Bar2DL2, 130
 - OFELI::Beam3DL2, 146
 - OFELI::DC1DL2, 185
 - OFELI::DC2DT3, 206
 - OFELI::DC2DT6, 223
 - OFELI::DC3DAT3, 243
 - OFELI::DC3DT4, 263
 - OFELI::EC2D1T3, 307
 - OFELI::Elas2DQ4, 331
 - OFELI::Elas2DT3, 351
 - OFELI::Elas3DH8, 367
 - OFELI::Elas3DT4, 382
 - OFELI::Equa_Electromagnetics, 404
 - OFELI::Equa_Fluid, 417
 - OFELI::Equa_Laplace, 430
 - OFELI::Equa_Solid, 445
 - OFELI::Equa_Therm, 459
 - OFELI::Equation, 472
 - OFELI::HelmholtzBT3, 506
 - OFELI::Laplace1DL2, 556
 - OFELI::Laplace1DL3, 569
 - OFELI::Laplace2DFVT, 582
 - OFELI::Laplace2DMHRT0, 595
 - OFELI::Laplace2DT3, 610
 - OFELI::NSP2DQ41, 714
 - OFELI::TINS2DT3B, 813
- setMatrix
 - OFELI::Eigen, 314, 315
 - OFELI::Prec, 731
- setMaxNbVertices
 - OFELI::MeshAdapt, 680
- setMesh

OFELI::SkMatrix, [750](#)
 OFELI::SkSMatrix, [766](#)
 OFELI::SpMatrix, [784](#)
 OFELI::SteklovPoincare2DBE, [796](#)
 OFELI::Vect, [845](#)
 setMeshFile
 OFELI::IOField, [533](#)
 setMethod
 OFELI::ICPG1D, [517](#)
 OFELI::ICPG2DT, [524](#)
 OFELI::ICPG3DT, [530](#)
 OFELI::LCL1D, [617](#)
 OFELI::LCL2DT, [623](#)
 OFELI::LCL3DT, [628](#)
 OFELI::Muscl, [684](#)
 OFELI::Muscl1D, [688](#)
 OFELI::Muscl2DT, [692](#)
 OFELI::Muscl3DT, [696](#)
 setN
 OFELI::Grid, [492](#)
 setNbDOFPerNode
 OFELI::Mesh, [668](#)
 setNoKeep
 OFELI::MeshAdapt, [680](#)
 setNoScaling
 OFELI::MeshAdapt, [680](#)
 setNodeBC
 OFELI::Vect, [846](#), [848](#)
 setNodeCodes
 Finite Element Mesh, [26](#)
 setNodeView
 OFELI::Mesh, [672](#)
 setOnBoundary
 OFELI::Node, [700](#)
 setPermeability
 OFELI::BiotSavart, [150](#)
 setPointInDomain
 OFELI::Mesh, [668](#)
 setPrintView
 OFELI::BMatrix, [163](#)
 OFELI::DMatrix, [273](#)
 OFELI::DSMatrix, [288](#)
 OFELI::Matrix, [658](#)
 OFELI::SkMatrix, [760](#)
 OFELI::SkSMatrix, [774](#)
 OFELI::SpMatrix, [794](#)
 OFELI::TrMatrix, [835](#)
 OFELI::Vect, [856](#)
 setRatio
 OFELI::MeshAdapt, [680](#)
 setReconstruction
 OFELI::ICPG1D, [517](#)
 OFELI::ICPG2DT, [522](#), [524](#)
 OFELI::ICPG3DT, [530](#)

OFELI::LCL1D, [617](#)
 OFELI::LCL2DT, [622](#)
 OFELI::LCL3DT, [628](#)
 OFELI::Muscl, [684](#)
 OFELI::Muscl1D, [688](#)
 OFELI::Muscl2DT, [691](#)
 OFELI::Muscl3DT, [695](#)
 setReferenceLength
 OFELI::ICPG1D, [517](#)
 OFELI::ICPG2DT, [524](#)
 OFELI::LCL2DT, [622](#)
 OFELI::Muscl, [684](#)
 OFELI::Muscl1D, [688](#)
 OFELI::Muscl2DT, [692](#)
 OFELI::Muscl3DT, [696](#)
 setRelaxation
 OFELI::MeshAdapt, [680](#)
 setRow
 OFELI::DMatrix, [269](#)
 setSide
 OFELI::Element, [390](#)
 setSideCodes
 Finite Element Mesh, [27](#)
 setSideView
 OFELI::Mesh, [673](#)
 setSize
 OFELI::BMatrix, [156](#)
 OFELI::DMatrix, [269](#)
 OFELI::DSMatrix, [286](#)
 OFELI::SpMatrix, [785](#)
 OFELI::TrMatrix, [828](#)
 OFELI::Vect, [845](#)
 setSkyline
 OFELI::SkMatrix, [751](#)
 OFELI::SkSMatrix, [766](#)
 setSolver
 OFELI::AbsEqua, [115](#)
 OFELI::Bar2DL2, [130](#)
 OFELI::Beam3DL2, [147](#)
 OFELI::DC1DL2, [186](#)
 OFELI::DC2DT3, [207](#)
 OFELI::DC2DT6, [224](#)
 OFELI::DC3DAT3, [244](#)
 OFELI::DC3DT4, [264](#)
 OFELI::EC2D1T3, [308](#)
 OFELI::Elas2DQ4, [332](#)
 OFELI::Elas2DT3, [352](#)
 OFELI::Elas3DH8, [367](#)
 OFELI::Elas3DT4, [382](#)
 OFELI::Equa_Electromagnetics, [404](#)
 OFELI::Equa_Fluid, [418](#)
 OFELI::Equa_Laplace, [431](#)
 OFELI::Equa_Solid, [445](#)
 OFELI::Equa_Therm, [460](#)

- OFELI::Equation, [472](#)
- OFELI::HelmholtzBT3, [507](#)
- OFELI::ICPG2DT, [522](#)
- OFELI::Laplace1DL2, [557](#)
- OFELI::Laplace1DL3, [570](#)
- OFELI::Laplace2DFVT, [583](#)
- OFELI::Laplace2DMHRT0, [595](#)
- OFELI::Laplace2DT3, [611](#)
- OFELI::NSP2DQ41, [715](#)
- OFELI::SpMatrix, [788](#)
- OFELI::TINS2DT3B, [814](#)
- setSource
 - OFELI::Laplace2DT3, [601](#)
- setStab
 - OFELI::DC1DL2, [177](#)
 - OFELI::DC2DT3, [198](#)
 - OFELI::DC2DT6, [215](#)
 - OFELI::DC3DAT3, [234](#)
 - OFELI::DC3DT4, [254](#)
 - OFELI::Equa_Therm, [451](#)
- setSubspaceDimension
 - OFELI::Eigen, [315](#)
- setTheta
 - OFELI::MeshAdapt, [681](#)
- setTimeStep
 - OFELI::ICPG1D, [517](#)
 - OFELI::ICPG2DT, [524](#)
 - OFELI::LCL1D, [616](#)
 - OFELI::LCL2DT, [622](#)
 - OFELI::LCL3DT, [628](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [696](#)
- setTolerance
 - OFELI::Eigen, [315](#)
- setTraction
 - OFELI::Laplace1DL2, [548](#)
 - OFELI::Laplace1DL3, [562](#)
- setTriangle
 - OFELI::Gauss, [486](#)
- setType
 - OFELI::Prec, [731](#)
- setVelocity
 - OFELI::LCL1D, [616](#)
 - OFELI::LCL2DT, [621](#), [622](#)
 - OFELI::LCL3DT, [627](#)
- setVerbose
 - OFELI::ICPG1D, [517](#)
 - OFELI::ICPG2DT, [524](#)
 - OFELI::ICPG3DT, [530](#)
 - OFELI::LCL1D, [617](#)
 - OFELI::LCL2DT, [623](#)
 - OFELI::LCL3DT, [628](#)
 - OFELI::Mesh, [667](#)
 - OFELI::Muscl, [684](#)
 - OFELI::Muscl1D, [688](#)
 - OFELI::Muscl2DT, [692](#)
 - OFELI::Muscl3DT, [696](#)
- setVertices
 - OFELI::Polygon, [728](#)
- setXMax
 - OFELI::Grid, [491](#)
- setXMin
 - OFELI::Grid, [491](#)
- Sh
 - OFELI::FEShape, [478](#)
 - OFELI::Hexa8, [509](#)
 - OFELI::Line2, [630](#), [631](#)
 - OFELI::Line2H, [633](#)
 - OFELI::Line3, [634](#)
 - OFELI::Penta6, [719](#)
 - OFELI::Quad4, [736](#)
 - OFELI::Tetra4, [799](#)
 - OFELI::Triang3, [817](#)
 - OFELI::Triang6S, [819](#), [820](#)
 - OFELI::triangle, [822](#), [824](#)
- Shape Function, [44](#)
- Side, [739](#)
 - OFELI::Side, [742](#)
- side_assembly
 - General Purpose Equations, [33](#), [34](#)
- SideAssembly
 - OFELI::Bar2DL2, [129](#)
 - OFELI::Beam3DL2, [144](#), [145](#)
 - OFELI::DC1DL2, [183](#), [184](#)
 - OFELI::DC2DT3, [204](#), [205](#)
 - OFELI::DC2DT6, [222](#), [223](#)
 - OFELI::DC3DAT3, [241](#), [242](#)
 - OFELI::DC3DT4, [261](#), [262](#)
 - OFELI::EC2D1T3, [305](#), [306](#)
 - OFELI::Elas2DQ4, [329](#), [330](#)
 - OFELI::Elas2DT3, [349](#), [350](#)
 - OFELI::Elas3DH8, [366](#)
 - OFELI::Elas3DT4, [381](#)
 - OFELI::Equa_Electromagnetics, [403](#)
 - OFELI::Equa_Fluid, [415](#), [416](#)
 - OFELI::Equa_Laplace, [428](#), [429](#)
 - OFELI::Equa_Solid, [444](#)
 - OFELI::Equa_Therm, [458](#), [459](#)
 - OFELI::Equation, [470](#), [471](#)
 - OFELI::HelmholtzBT3, [504](#), [505](#)
 - OFELI::Laplace1DL2, [554](#), [555](#)
 - OFELI::Laplace1DL3, [567](#), [568](#)
 - OFELI::Laplace2DFVT, [580](#), [581](#)
 - OFELI::Laplace2DMHRT0, [594](#)
 - OFELI::Laplace2DT3, [608](#), [609](#)
 - OFELI::NSP2DQ41, [713](#), [714](#)

OFELI::TINS2DT3B, [812](#), [813](#)
 SideInElement
 Finite Element Mesh, [29](#)
 SideList, [745](#)
 SideNodeCoordinates
 OFELI::Bar2DL2, [126](#)
 OFELI::Beam3DL2, [142](#)
 OFELI::DC1DL2, [181](#)
 OFELI::DC2DT3, [202](#)
 OFELI::DC2DT6, [219](#)
 OFELI::DC3DAT3, [239](#)
 OFELI::DC3DT4, [259](#)
 OFELI::EC2D1T3, [303](#)
 OFELI::Elas2DQ4, [327](#)
 OFELI::Elas2DT3, [347](#)
 OFELI::Elas3DH8, [362](#)
 OFELI::Elas3DT4, [377](#)
 OFELI::Equa_Electromagnetics, [400](#)
 OFELI::Equa_Fluid, [413](#)
 OFELI::Equa_Laplace, [426](#)
 OFELI::Equa_Solid, [440](#)
 OFELI::Equa_Therm, [455](#)
 OFELI::Equation, [468](#)
 OFELI::HelmholtzBT3, [502](#)
 OFELI::Laplace1DL2, [552](#)
 OFELI::Laplace1DL3, [565](#)
 OFELI::Laplace2DFVT, [578](#)
 OFELI::Laplace2DMHRT0, [591](#)
 OFELI::Laplace2DT3, [606](#)
 OFELI::NSP2DQ41, [710](#)
 OFELI::TINS2DT3B, [809](#)
 SideType
 OFELI::Side, [742](#)
 SideVector
 OFELI::Bar2DL2, [126](#)
 OFELI::Beam3DL2, [141](#)
 OFELI::DC1DL2, [180](#)
 OFELI::DC2DT3, [201](#)
 OFELI::DC2DT6, [219](#)
 OFELI::DC3DAT3, [238](#)
 OFELI::DC3DT4, [258](#)
 OFELI::EC2D1T3, [302](#)
 OFELI::Elas2DQ4, [326](#)
 OFELI::Elas2DT3, [346](#)
 OFELI::Elas3DH8, [362](#)
 OFELI::Elas3DT4, [377](#)
 OFELI::Equa_Electromagnetics, [400](#)
 OFELI::Equa_Fluid, [412](#)
 OFELI::Equa_Laplace, [425](#)
 OFELI::Equa_Solid, [440](#)
 OFELI::Equa_Therm, [455](#)
 OFELI::Equation, [468](#)
 OFELI::HelmholtzBT3, [501](#)
 OFELI::Laplace1DL2, [551](#)

OFELI::Laplace1DL3, [564](#)
 OFELI::Laplace2DFVT, [577](#)
 OFELI::Laplace2DMHRT0, [591](#)
 OFELI::Laplace2DT3, [605](#)
 OFELI::NSP2DQ41, [710](#)
 OFELI::TINS2DT3B, [809](#)
 SidesAreDOF
 OFELI::Mesh, [675](#)
 SignoriniContact
 OFELI::Elas2DQ4, [322](#)
 OFELI::Elas2DT3, [341](#)
 SkMatrix
 OFELI::SkMatrix, [750](#)
 SkMatrix< T_ >, [746](#)
 SkSMatrix
 OFELI::SkSMatrix, [764](#), [765](#)
 SkSMatrix< T_ >, [761](#)
 Solid Mechanics, [45](#)
 Solve
 OFELI::BMatrix, [157](#), [161](#)
 OFELI::DMatrix, [271](#), [272](#)
 OFELI::DSMatrix, [288](#)
 OFELI::LocalMatrix, [640](#)
 OFELI::Matrix, [655](#)
 OFELI::Prec, [731](#)
 OFELI::SkMatrix, [754](#), [755](#)
 OFELI::SkSMatrix, [768](#), [769](#)
 OFELI::SpMatrix, [787](#), [788](#)
 OFELI::SteklovPoincare2DBE, [796](#)
 OFELI::TrMatrix, [829](#), [833](#)
 solve
 OFELI::Laplace2DMHRT0, [588](#)
 OFELI::Laplace2DT3, [602](#)
 solveEigenProblem
 OFELI::AbsEqua, [116](#)
 OFELI::Bar2DL2, [131](#)
 OFELI::Beam3DL2, [147](#)
 OFELI::DC1DL2, [186](#)
 OFELI::DC2DT3, [207](#)
 OFELI::DC2DT6, [225](#)
 OFELI::DC3DAT3, [244](#)
 OFELI::DC3DT4, [264](#)
 OFELI::EC2D1T3, [308](#)
 OFELI::Elas2DQ4, [332](#)
 OFELI::Elas2DT3, [352](#)
 OFELI::Elas3DH8, [368](#)
 OFELI::Elas3DT4, [383](#)
 OFELI::Equa_Electromagnetics, [405](#)
 OFELI::Equa_Fluid, [418](#)
 OFELI::Equa_Laplace, [431](#)
 OFELI::Equa_Solid, [446](#)
 OFELI::Equa_Therm, [461](#)
 OFELI::Equation, [473](#)
 OFELI::HelmholtzBT3, [507](#)

- OFELI::Laplace1DL2, 557
- OFELI::Laplace1DL3, 570
- OFELI::Laplace2DFVT, 583
- OFELI::Laplace2DMHRT0, 596
- OFELI::Laplace2DT3, 611
- OFELI::NSP2DQ41, 716
- OFELI::TINS2DT3B, 815
- Solver, 46
 - BCAsConstraint, 54
 - BiCG, 49
 - BiCGStab, 50
 - CG, 51
 - CGS, 51
 - GMRes, 52
 - GS, 53
 - IterationLoop, 48
 - Jacobi, 53
 - MAX_NB_EQUATIONS, 48
 - MAX_NB_INPUT_FIELDS, 48
 - MAX_NB_MESHES, 48
 - OptimSA, 54
 - OptimTN, 57
 - QMR, 58
 - Richardson, 59
 - SSOR, 61
 - Schur, 60
 - TimeLoop, 48
- SolverType
 - OFELI::ICPG1D, 513
 - OFELI::ICPG2DT, 521
 - OFELI::ICPG3DT, 528
 - OFELI::LCL1D, 615
 - OFELI::LCL2DT, 620
 - OFELI::LCL3DT, 626
 - OFELI::Muscl, 683
 - OFELI::Muscl1D, 687
 - OFELI::Muscl2DT, 691
 - OFELI::Muscl3DT, 695
- SpMatrix
 - OFELI::SpMatrix, 782–784
- SpMatrix< T_ >, 778
- Sphere, 775
 - OFELI::Sphere, 776
- Start
 - OFELI::Timer, 801
- Started
 - OFELI::Timer, 801
- SteklovPoincare2DBE, 794
 - OFELI::SteklovPoincare2DBE, 795
- Stiffness
 - OFELI::Bar2DL2, 122
- Stop
 - OFELI::Timer, 801
- Strain
 - OFELI::Elas2DQ4, 322
 - OFELI::Elas2DT3, 343
- Stress
 - OFELI::Elas2DQ4, 323
 - OFELI::Elas2DT3, 343
- SurfaceForce
 - OFELI::UserData, 836
- Symmetrize
 - OFELI::LocalMatrix, 640
- TEMPERATURE_FIELD
 - OFELI, 109
- THERMAL_PHASE_CHANGE
 - OFELI, 109
- TRACTION
 - OFELI, 108
- TRANSIENT
 - OFELI, 110
- TRIDIAGONAL
 - OFELI, 110
- TINS2DT3B, 801
 - OFELI::TINS2DT3B, 805
- TMult
 - OFELI::DMatrix, 270
 - OFELI::DSMatrix, 287
 - OFELI::SkMatrix, 752
 - OFELI::SkSMatrix, 767
 - OFELI::SpMatrix, 786
- TMultAdd
 - OFELI::SkMatrix, 752
- Tabulation, 796
- Tetra4, 798
- TheEdge
 - Finite Element Mesh, 21
- theEdge
 - Global Variables, 36
- TheElement
 - Finite Element Mesh, 21
- theElement
 - Global Variables, 35
- theFinalTime
 - Global Variables, 37
- theliteration
 - Global Variables, 36
- TheNode
 - Finite Element Mesh, 21
- theNode
 - Global Variables, 35
- theNodeLabel
 - Finite Element Mesh, 23
- TheSide
 - Finite Element Mesh, 21
- theSide
 - Global Variables, 36

- theStep
 - Global Variables, [36](#)
- theTime
 - Global Variables, [37](#)
- theTimeStep
 - Global Variables, [36](#)
- theTolerance
 - Global Variables, [37](#)
- TimeLoop
 - Solver, [48](#)
- TimeScheme
 - OFELI, [109](#)
- Timer, [800](#)
- TrMatrix
 - OFELI::TrMatrix, [828](#)
- TrMatrix< T_ >, [825](#)
- TransSolve
 - OFELI::Prec, [731](#), [732](#)
- transferBC
 - OFELI::Vect, [849](#)
- Triang3, [815](#)
 - OFELI::Triang3, [817](#)
- Triang6S, [818](#)
 - OFELI::Triang6S, [819](#)
- triangle, [820](#), [822](#)
 - OFELI::triangle, [822](#), [824](#)
- UNSYMMETRIC
 - OFELI, [110](#)
- unselectCode
 - OFELI::EdgeList, [312](#)
 - OFELI::ElementList, [392](#)
 - OFELI::NodeList, [702](#)
 - OFELI::SideList, [746](#)
- updateBC
 - OFELI::Bar2DL2, [123](#)
 - OFELI::Beam3DL2, [139](#)
 - OFELI::DC1DL2, [178](#)
 - OFELI::DC2DT3, [199](#)
 - OFELI::DC2DT6, [216](#), [217](#)
 - OFELI::DC3DAT3, [236](#)
 - OFELI::DC3DT4, [255](#)
 - OFELI::EC2D1T3, [300](#)
 - OFELI::Ela2DQ4, [324](#)
 - OFELI::Ela2DT3, [344](#)
 - OFELI::Ela3DH8, [359](#), [360](#)
 - OFELI::Ela3DT4, [374](#), [375](#)
 - OFELI::Equa_Electromagnetics, [397](#)
 - OFELI::Equa_Fluid, [409](#)
 - OFELI::Equa_Laplace, [422](#)
 - OFELI::Equa_Solid, [437](#), [438](#)
 - OFELI::Equa_Therm, [452](#), [453](#)
 - OFELI::Equation, [465](#)
 - OFELI::HelmholtzBT3, [499](#)
 - OFELI::Laplace1DL2, [549](#)
 - OFELI::Laplace1DL3, [562](#)
 - OFELI::Laplace2DFVT, [575](#)
 - OFELI::Laplace2DMHRT0, [588](#)
 - OFELI::Laplace2DT3, [603](#)
 - OFELI::NSP2DQ41, [707](#), [708](#)
 - OFELI::TINS2DT3B, [805](#), [807](#)
- UserData< T_ >, [835](#)
- Utilities, [63](#)
 - Axpy, [82](#)
 - BSpline, [80](#)
 - banner, [81](#)
 - Clear, [82](#)
 - Dot, [82](#)
 - EVAL, [70](#)
 - Equal, [83](#)
 - getBamg, [75](#)
 - getEasymesh, [76](#)
 - getGambit, [76](#)
 - getGmsh, [76](#)
 - getMatlab, [77](#)
 - getMesh, [74](#)
 - getNetgen, [77](#)
 - getTetgen, [77](#)
 - getTriangle, [78](#)
 - Nrm2, [82](#)
 - OFELI_E, [70](#)
 - OFELI_EPSMCH, [70](#)
 - OFELI_GAUSS2, [70](#)
 - OFELI_IMAG, [70](#)
 - OFELI_ONEOVERPI, [70](#)
 - OFELI_PI, [70](#)
 - OFELI_SIXTH, [70](#)
 - OFELI_SQRT2, [70](#)
 - OFELI_SQRT3, [70](#)
 - OFELI_THIRD, [70](#)
 - OFELI_TOLERANCE, [70](#)
 - OFELI_TWELVETH, [70](#)
 - operator*, [72–74](#)
 - operator+, [72](#), [73](#)
 - operator-, [72](#), [73](#)
 - operator/, [73](#), [74](#)
 - operator==, [72](#), [73](#)
 - PARSE, [70](#)
 - qksort, [81](#)
 - QuickSort, [81](#)
 - saveBamg, [80](#)
 - saveField, [71](#)
 - saveGmsh, [72](#), [79](#)
 - saveGnuplot, [71](#), [79](#)
 - saveMatlab, [79](#)
 - saveMesh, [78](#)
 - saveTecplot, [71](#), [79](#)
 - saveVTK, [72](#), [80](#)

- Scale, [82](#)
- Xpy, [82](#)
- Xpy Utilities, [82](#)
- VANALBADA_LIMITER
 - OFELI::ICPG1D, [513](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- VANLEER_LIMITER
 - OFELI::ICPG1D, [513](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- VELOCITY_FIELD
 - OFELI, [109](#)
- VFROE_SOLVER
 - OFELI::ICPG1D, [514](#)
 - OFELI::ICPG2DT, [521](#)
 - OFELI::ICPG3DT, [528](#)
 - OFELI::LCL1D, [615](#)
 - OFELI::LCL2DT, [620](#)
 - OFELI::LCL3DT, [626](#)
 - OFELI::Muscl, [683](#)
 - OFELI::Muscl1D, [687](#)
 - OFELI::Muscl2DT, [691](#)
 - OFELI::Muscl3DT, [695](#)
- VISCOSITY
 - OFELI, [108](#)
- Vect
 - OFELI::Vect, [841–844](#)
- Vect< T_ >, [837](#)
- Vector and Matrix, [84](#)
 - Dot, [87](#)
 - operator*, [86](#), [87](#)
 - operator+, [86](#)
 - operator-, [86](#), [87](#)
 - operator/, [86](#), [87](#)
- Verbosity
 - Global Variables, [36](#)
- WithMesh
 - OFELI::Vect, [846](#)